Cruzr SDK Integration Guide

Revision History				
Date	Version	Revised by	Description	Remarks
2017.01.03	v0.1	Zhang Junjian	First draft of the documentation on interfaces	
2017.01.16	v0.1	Hu Yingqi	Added the facial recognition engine interface	
2017.02.16	v0.1	Zhang Junjian	Added offline command issuance	
2017.03.08	v0.1	Yang Ning	Added two parameters to the facial expressions interface	
2017.03.10	v0.2	Wang Jiajin	Added a method to upload catchphrases to the speech interface	
2017.03.14	v0.3	Wang Jiajin	Added the voice command callback interface, voice dictation callback interface, TTS callback interface, and the rule that SpeechService must be declared when registering the voice command callback	
2017.06.17	v0.4	Zhang Junjian	Moved interface instructions to documentation, and provided integration instructions in the following part of this document	
2017.07.17	v0.4	Zhang Junjian	Added the methods for setProperty and getProperty	
2017.09.07	v0.5	Dan Lili	Added instructions on command configuration, ADB debugging, API usage, and facial recognition API	
2018.01.18	v0.6	Zhao Zekai	Improved the ROS and speech parts	
2018.04.10	v0.7	Zhang Junjian	Added navigation, dance, and facial recognition interfaces, FAQs, and modified API documentation	Version 0.1.8

Contents

1. Document Overview 3
1.1 Functional Description 3
1.2 Target Audience 3
2. Preparations 3
2.1. Eclipse Environment 3
2.2 Android Studio Environment 3
3. Code Integration 5
3.1 ROS Control Engine Integration 5
3.1.1 Initialization 5
3.1.2 Joint control 5
3.1.3 Map operation 6
3.1.4 Charging operation 6
3.1.5 Navigation operation 6
3.1.6 Move operation 7
3.1.7 Diagnosis operation 7
3.1.8 Power operation 7
3.1.9 LED light 8
3.1.10 Alert & alarm 8
3.1.11 General information collection 8
3.1.12 Preset action playback 9
3.2 Speech Engine Integration 11
3.2.1 Initialization 11
3.2.2 TTS 12
3.2.3 ASR 12
3.2.4 Voice command issuing 13
3.2.5 Speech Engine Replacement 15
3.3 State Machine Integration 21
3.4 Navigation Control Integration 21
3.4.1 Initialization 21
3.4.2 Registering event callback 21
3.4.3 Obtaining the current map 22
3.4.4 Setting up the current map 23
3.4.5 Starting map positioning 24
3.4.6 Canceling map positioning 24
3.4.7 Querying all map points 25
3.4.8 Starting and canceling navigation 26
3.4.9 Obtaining the current location 26
3.5 Facial Recognition Integration 27
3.5.1 Initialization 28
3.5.2 Still image detection 29
3.5.3 Image feature extraction 29
3.6 Facial Expression Integration 30
3.7 Dance Integration 30
4. ADB Debugging 32
5. FAQs 32

1. Document Overview

1.1 Functional Description

This document is the integration guide for developing UBTECH robot applications on third-party platforms. Developers can download the corresponding SDK on the official website. We provide functions including basic motions, motion control, voice dictation, speech synthesis, offline commands, facial recognition, expression control, navigation control, state machine, and dance control. This document mainly describes the SDK integration method so that developers can quickly start the SDK integrated development on Android. For interface details, please see the appendix. Every time our system is updated, the corresponding SDK version will be released. Third-party developers can contact our technical personnel at any time if there are any questions during the usage of the SDK.

1.2 Target Audience

This document is intended for developers who are capable of Android development and administrators who understand Android development.

2. Preparations

2.1. Eclipse Environment

Download the SDK and copy it to the "libs" directory. If no such folder exists, create one as shown in Figure 2-1.

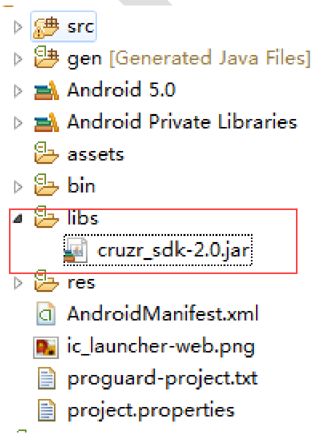

```
src
gen [Generated Java Files]
Android 5.0
Android Private Libraries
assets
bin
libs
    cruzr_sdk-2.0.jar
res
AndroidManifest.xml
ic_launcher-web.png
proguard-project.txt
project.properties
```

Figure 2-1 Import SDK

2.2 Android Studio Environment

Download the SDK and copy it to the "libs" directory. If no such folder exists, create one as shown in Figure 2-2.

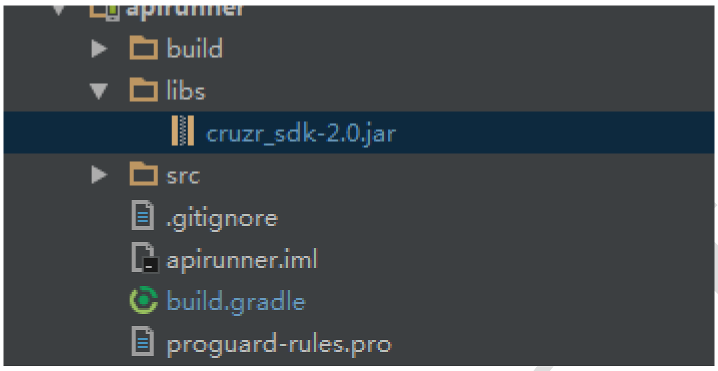


Figure 2-2 Import SDK

```
apply plugin: 'com.android.application'

android {
        compileSdkVersion 21
        buildToolsVersion 21
        defaultConfig {
                applicationId "com.ubtechinc.sdk.demo"
                minSdkVersion 21
                targetSdkVersion 21
                versionCode 1
                versionName "1.0"
        }

        buildTypes {
                release {
                        signingConfig signingConfigs.config
                        minifyEnabled false
                        proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
                }
        }
        sourceSets {
                main {
                        jniLibs.srcDirs = ['libs']
                }
        }
}

dependencies {
        compile files('libs/cruzr_sdk-2.0.jar')
}
```

3. Code Integration

3.1 ROS Control Engine Integration

The ROS control engine mainly provides support for movement, motion control, LED light effect control, and system notifications. Third-party developers can control basic functions of the robot such as moving forward/backward, turning left/right, raising the head, system actions, and dancing using this interface.

ROS interfaces can be classified into the following types:

- Joint control
- Map operation
- Charging operation
- Navigation operation
- Move operation
- Diagnosis operation
- Power operation
- LED light
- Alert & alarm
- General information collection
- Preset action playback

3.1.1 Initialization

The initialization API should be called from the launching Application of the app. You can use other interfaces only after a successful callback is received. Here is the example code:

```
RosRobotApi.get().initializ(this, new InitListener() {
                    @Override
                    public void onInit() {
                        //Initialization successful
                    }
                });
```

3.1.2 Joint control

The joint control interface allows controls of the upper and lower head joints, left and right wrist joints, and left and right elbow joints. See the documentation for detailed interface instructions, and refer to the demo for the method of use. Here are the relevant interfaces:

```
public int setAngles(java.lang.String[] names, float[] angles, float[] times)
```

```
public int setAngles(java.lang.String name, float angle, float time)
```

public int stopJoint()

public float getJointCurrentAngle(java.lang.String name)

public int angleInterpolation(java.lang.String[] names, java.util.List<float[]> angles, java.util.List<float[]> times, int absolute) (Reserved)

public int angleInterpolationWithSpeed(java.lang.String[] names, java.util.List<float[]> angles, float max_speed, int absolute) (Reserved)

Note: Currently, all the motion scope angles in the API are measured in radians. The value range is (-pi, pi).

3.1.3 Map operation

The map interface allows syncing of the maps to the ROS. A detailed description on the usage of map and navigation-related integration can be found in 3.4 Navigation Control Integration. See the documentation for detailed interface instructions, and refer to the demo for the method of use. Here are the relevant interfaces:

public int setCurrentMap(java.lang.String name)

public java.lang.String getCurrentMap()

public int clearMap()

3.1.4 Charging operation

The charging interface allows the robot to go to the charging dock at the specified coordinates to charge. Developers write in the coordinates and make the robot navigate to the specified coordinates before connecting with the charging dock. See the documentation for detailed interface instructions, and refer to the demo for the method of use. Here are the relevant interfaces:

public int dockOnStation()

public int cancelDockOn()

public int gotoRecharge(float x,float y)

public int leaveStation()

3.1.5 Navigation operation

The navigation control interface allows developers to specify the coordinates for the robot to accurately navigate to. The relevant interfaces are system built-in capabilities, and we have provided secondary encapsulation for developers to use based on the actual map and navigation usage. For details, go to Section 3.4. See the documentation for detailed interface instructions, and refer to the demo for the method of use. Here are the relevant interfaces:

public int navigateToByPresetedSpeed(float x, float y, float theta)

```
public int navigateTo(float x, float y, float theta, float maxSpeed)

public int cancelNavigate()

public int navigateRelocationCtrl(int opt)

public int navigateRelocationStartByPos(float x, float y, float theta)
```

3.1.6 Move operation

The movement control interface allows developers to control the movement of the robot, including moving forward and backward, turning left and right, etc. Unlike navigation, real world coordinates are used here. See the documentation for detailed interface instructions, and refer to the demo for the method of use. Here are the relevant interfaces:

```
public int moveTo(float x, float y, float theta, float maxSpeed)

public int moveTo(float x, float y, float theta, float maxSpeed, RemoteCommonListener listener)

public int moveToward(float x,float y, float theta)

public int moveToward(float x,float y,float theta,RemoteCommonListener listener)

public int stopMove()

public boolean isMoveActive()

public Position getPosition(boolean useSensor)
```

3.1.7 Diagnosis operation

The diagnosis control interface allows diagnosis on the functioning of system sensors. This interface is usually for system internal use. See the documentation for detailed interface instructions, and refer to the demo for the method of use. Here are the relevant interfaces:

```
public int diagnosticDataReportCtrl(int key, int ctrlflag)

public int getDiagnosticStatus(int key)

public int registerDiagnosticDataCallback(RemoteDiagnosticDataListener listener)
```

Note: See Appendix A for specific module ID

3.1.8 Power operation

The power control interface allows the control of the ROS power, usually operations executed by administrators with system permissions. See the documentation for detailed interface instructions, and refer to the demo for the method of use. Here are the relevant interfaces:

public int powerOn(String currentTime, String scheduleTime, String option)

public int powerOffRos(int type)

public int registerPowerCallback(RemotePowerListener listener)

3.1.9 LED light

The LED light interface allows the control of the light effect, brightness, and color of the LED light panel by the ROS. See the documentation for detailed interface instructions, and refer to the demo for the method of use. Here are the relevant interfaces:

public int ledSetColor(int r, int g, int b, int times)

public int ledSetEffect(int lightEffect, int brightness, int color)

public int ledSetWorkByTimes(int effect, int brightness, int color, int times)

public int ledSetOnOff(boolean onOff)

3.1.10 Alert & alarm

The ROS API is generally engaged in asynchronous operations. Every time a movement or action interface is called, there will be a common callback. Developers can register a common listener and compare the callback interface sessionid with the return value of the call method to determine if it is the current call. For alerts and alarms, register a warn listener, and any system events will be called back through this interface. See the documentation for detailed interface instructions, and refer to the demo for the method of use. Here are the relevant interfaces:

public int registerCommonCallback(RemoteCommonListener listener)

public int registerWarnCallback(RemoteWarnListener listener)

3.1.11 General information collection

The general information collection interface provides a synchronous call method. See the documentation for detailed interface instructions, and refer to the demo for the method of use. Here are the relevant interfaces:

public BatteryInfo getBattery()

public float[] getHumiture()

public java.lang.String getRosVersion()

public java.lang.String[] getVersionInfo()

3.1.12 Preset action playback

The action playback interface allows developers to control the robot's actions. Basic actions are built into the system. Developers can simply pass the action name through the interface to carry out the basic action function. Currently, actions are preset in the system and do not support customization. To customize actions, please contact the Cruzr product team. See the documentation for detailed interface instructions, and refer to the demo for the method of use. Here are the relevant interfaces:

```
public int run(java.lang.String name)

public int run(java.lang.String[] names)

public int stopRun()
```

Here is the current list of basic actions, which will be updated later:

Action	Name	Description
Salute	pose1	Salutes
Gesticulate	pose2	Makes a gesture with both hands in front of chest
Whatever	pose3	Shrugs shoulders to show indifference
Hug	hug	Hugs
Handshake	shankhand	Reaches out to shake hands
Goodbye	goodbye	Waves a hand to say goodbye
Talk 1	talk1	Raise right hand (used in speeches)
Talk 2	talk2	Raise left hand (used in speeches)
Talk 3	talk3	Raise both hands (used in speeches)
Talk 5	talk5	Moves both hands
Talk 6	talk6	Moves left hand
Talk 7	talk7	Moves right hand
Talk 8	talk8	Up and down

Nod	nod	Nods head
Clap	applause	Claps hands
Scratch head	zhuatou	Scratches head when confused
Walk right	guideright	Guides visitors to walk to the right
Walk left	guideleft	Guides visitors to walk to the left
Cute	cute	If you make the robot say "Hmph! How dare you!", the effect will be better
Fight	fendou	Shows fighting appearance
Grow	zhanggao	Cruzr acts like it wants to get taller
Swing arms	swingarm	Swings both arms
Search	searching	Turns body and looks up and down
Zone out	fadai	Turns body and raises/lowers head
Survey	tiaowang	Raises hand to head and looks left to right
Grab	longzhuashou	Raises one hand and makes a grabbing gesture
Protect	baohu	Raises both hands to protect itself
Wonder	surprise	Raises both hands
Guide	zhilu	Raises right hand
Shy	shy	Covers face and sways hips to show shyness
Conduct	command	Acts as if conducting music
End conducting	commandover	Finalizes the conducting action
Walk	walk	Waves hand
Reset	reset	Resets (to the initial default status of the robot)

3.2 Speech Engine Integration

The voice interaction engine mainly allows third-party developers to integrate speech recognition, speech synthesis, and command issuing functions. Both offline commands and online commands can be issued. Offline commands are configured via developer coding before being scanned by the command system and built into the command set, while online custom commands require registering an enterprise account and CBI dynamic configuration.

3.2.1 Initialization

This is called from the launching Application of the app. You can use speech-related interfaces only after a successful callback is received. Here is the example code:

```
SpeechRobotApi.get().initializ(this, 103,new InitListener() {
                    @Override
                    public void onInit() {
                        //Initialization successful
                    }
                });
```

Note: To get an appid, you need to contact Cruzr staff to apply for a unique ID.
Note: Command issuing refers to the process of issuing speech recognition results to third-party applications.

Offline command configuration: Simply configure the commands to be processed on the local app to connect to the system issuing platform. This is currently how a third-party connection is usually established. Here are the configuration steps:

1) Add the "meta-data" label under "application" in AndroidManifest.xml to configure the issuing, wherein value is the appid provided by the Cruzr team. Third-party developers can contact the Cruzr team to apply for this ID if needed.

```
<meta-data
     android:name="cruiser_appid"
     android:value="103" />
```

2) Add your own offline commands in the following format to "values" under res of the application. The system will scan all commands defined by cruiser_call during startup and build them into the voice service command set. Currently, only fully matched commands will be issued, so developers can provide more commands to increase the rate of successful recognition and issuing. Here is an example:

```
<string name="cruiser_call">Come|Come over|Please come over|Can you come over|Could you come over|Would you come here for a second|Please come|Please come over here</string>
```

3) Register related services under "application" in AndroidManifest.xml.

```
<service
     android:name="com.ubtechinc.cruzr.sdk.speech.SpeechService"
     android:enabled="true"
     android:exported="true">
    <intent-filter>
        <action android:name=" Your package name.Cruiser_Service"/>
    </intent-filter>
```

```
</service>
```

Note: The action content in the above code must be package name + ".Cruiser_Service". Please make sure the correct package name is used, and note that ".Cruiser_Service" is case sensitive.

Online command configuration:

This is currently not open to third-party developers. You can contact the Cruzr product team CBI to configure custom online commands if needed.

3.2.2 TTS

TTS is the system function that enables third-party broadcasting. Currently there are 3 Chinese voice sources and one English voice source to choose from. To change the voice source, please select under Settings > Voice System. See the documentation for detailed interface instructions, and refer to the demo for the method of use. Here are the relevant interfaces:

```
public int speechSetTtsVolume(int volume)

public int speechStartTTS(String text, SpeechTtsListener listener)

public int speechStopTTS()

public List<SpeechVoice> getSpeechVoices()

public SpeechVoice getCurSpeechVoices()

public int speechGetTtsSpeed()

public boolean isTtsSpeaking()

public String speechGetVoiceName()
```

3.2.3 ASR

Automatic Speech Recognition (ASR) is the system function that enables third-party speech recognition. Developers do not need to integrate a speech engine. By simply calling the interface, developers can obtain the speech recognition content for the last 15 seconds (default timeout). Currently, ASR only supports Chinese and English. See the documentation for detailed interface instructions, and refer to the demo for the method of use. Here are the relevant interfaces:

```
public int startSpeechASR(SpeechASRListener listener)

public int stopSpeechASR()
```

Note: The ASR function will keep calling back the recognized content before timeout. If developers have already received the correct recognition content, please call the "stop" interface to stop ASR, otherwise it will not stop until timeout.

3.2.4 Voice command issuing

This function allows the system to issue voice commands to third-party developers. Developers can configure a unique appid and command set through online or offline means. When being waked, the system will try to match offline commands based on currently recognized content, and issue the command if a successful match is found. Otherwise, the system will start the online command process. Both online and offline commands are in json format on the developers' terminal. The standard format also includes distinctions between online and offline commands, wakeup mode, recognition content and other information (see Note for the format example). The system also allows developers to configure the interface so that all commands or all commands except offline commands are issued to the specified appid application. See the documentation for detailed interface instructions, and refer to the demo for the method of use. Here are the relevant interfaces:

```
public int speechPermissionDispatch(int appId, int permission)

public int enableWakeup(int wakeType, boolean enable)

SpeechRobotApi.get().registerSpeech(new ISpeechContext() {
                    @Override
                    public void onStart() {
                    }

                    @Override
                    public void onStop() {
                    }

                    @Override
                    public void onResult(String s) {
                        Log.i("dan", "support onResult:" + s);
                        //Third parties receive a json command issued by the system
                    }

                    @Override
                    public void onPause() {
                    }

                    @Override
                    public void onResume() {
                    }
                });
```

Note: System voice wakeup, button wakeup, and vision wakeup all broadcast notifications. Developers can listen to the notifications as required, or disable the wakeup functions through the interface. Here is the relevant information:

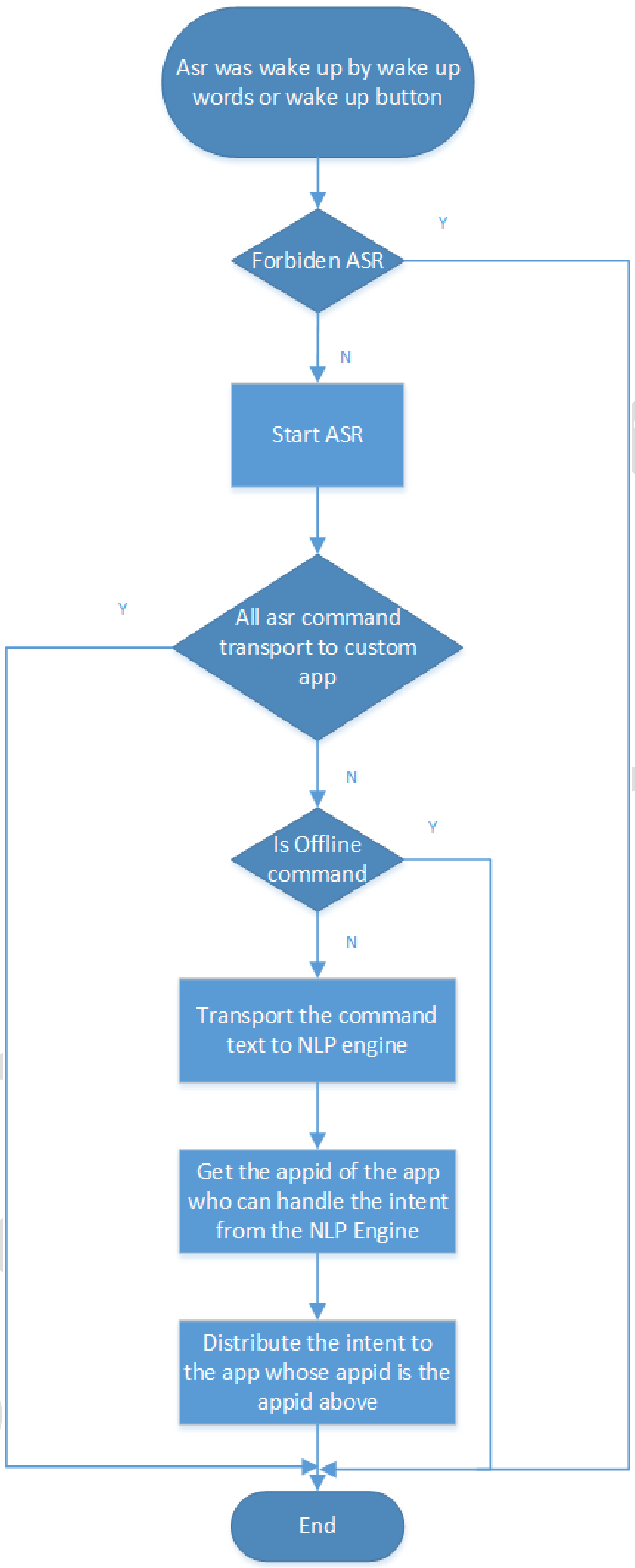
```
//Third-party wakeup broadcasts, including voice wakeup, virtual button wakeup, and vision wakeup
public static final String CORE_SERVICE_WAKE_UP = "com.ubtechinc.cruzr.coreservices.ACTION.WAKE_UP";
public static final String CRUISER_DATA_WAKE_UP_TYPE = "wakeup_type";
```

```
//Wakeup type definition
public static final int WAKE_UP_TYPE_SPEECH = 0;
public static final int WAKE_UP_TYPE_KEY = 1;
public static final int WAKE_UP_TYPE_FACE_IN = 2;
public static final int WAKE_UP_TYPE_FACE_OUT = 3;
```

Note: Command issuing format

```
{
        //Command issuing appid
        "appid": 13,
        //Default
        "id": 0,
         //0: offline command; 1: online command
        "command": 0,
        //Voice intent
        "intent": "Dance",
        "meta": "",
        "reply": "",
         //Speech recognition content
        "request": "Do a dance",
         //Wakeup method - 0: voice wakeup; 1: virtual button wakeup; 2: vision wakeup
        "wakeup": 2
}
```

Command issuing workflow chart:


Asr was wake up by wake up words or wake up button
Forbiden ASR
Y
N
Start ASR
All asr command transport to custom app
Y
N
Is Offline command
Y
N
Transport the command text to NLP engine
Get the appid of the app who can handle the intent from the NLP Engine
Distribute the intent to the app whose appid is the appid above
End

3.2.5 Speech Engine Replacement

We allow third-party developers to replace the system speech and semantic engines. The system speech and semantic engines use an open and adaptive design and third-party users can replace it as needed. The current default speech and semantic engines are both "ubt". Third-party applications can customize the engine configuration and use the "ubt" engine with other engines. See the following for detailed integration instructions.

1) Configuration method

Use ContentProvider to configure the voice system engine, wherein
android:authorities="com.ubtechinc.speech.cfg.provider".
The voice system will scan the ContentProvider automatically during startup, and obtain the configuration content through **Bundle call(String method, String arg, Bundle extras)**,
where the method parameter is **"readSpeechCfg"**, and the return value Bundle contains String type information as **bl.putString("config", mConfig)**.
If ContentProvider does not exist or has not been successfully obtained, the voice system will use the default speech engine.

```
<provider
      android:authorities="com.ubtechinc.speech.cfg.provider"
      android:name="com.ubtechinc.speechservice.demo.TestProvider"
      android:exported="true" />

 @Override
public Bundle call(String method, String arg, Bundle extras) {
      if (!TextUtils.isEmpty(method) && method.equals("readSpeechCfg")) {
            Bundle bl = new Bundle();
            bl.putString("config", mConfig);
            return bl;
      }
      return super.call(method, arg, extras);
}
```

2) Configuration content

See the "speech.cfg" file under "demo" for the configuration file.

```
{
    "voiceEngine" :{
        "engine": "ubt",
        "nlp": "ubt",
                  "supportContinue": ["zh-CN","zh-TW"]
   },

   "voiceTts" :
     [
        {
            "lan" : "zh-CN",
            "voice": "ybxf1",
            "speed": "50",
            "tts":
             [
                    {
                        "name": "ybxf1",
                                                            "sex": 1,
                                                            "adult": 1
                                                    },
                                                    {
                                                                "name": "xiaoyan",
                                                                "sex": 1,
                                                                "adult": 1
                                                        },
                                                    {
                                                            "name": "xiaofeng",
                                                            "sex": 0,
                                                            "adult": 1
                                                    }
                                            ]
                                    }
                            ]
                    }
```

The configuration content is strings in JSON format. The fields after resolution are shown as follows:

```
            public class SpeechCfgs {
                                    //Speech recognition and semantic configuration information
                        public VoiceEngine voiceEngine;
                                    //Speech pronunciation configuration information
                        public List<VoiceTts> voiceTts;

                        public static class VoiceEngine {
                                    //Speech recognition engine key. Use this key to match the value of the speech engine
(meta-data name ="cruiser_speech")
                                    public String engine;
                                    //Semantic recognition engine key. Use this key to match the value of the semantic engine
(meta-data name ="cruiser_nlp")
                                    public String nlp;
                                    //If the default speech engine "ubt" is used, you can configure the function of continuous
recognition. Currently only "zh-CN" and "zh-TW" are supported.
                                    //This data is invalid for third-party engines
                                    public List<String> supportContinue;
                        }

            public static class VoiceTts implements IProguardKeeper {
                                    //Current language
                                    public String lan;
                                    //Default voice source
                                    public String voice;
                                    //Default voice speed
                                    public int speed;
                                    //List of voice sources supported by the current language
                                    public List<SpeechVoice> tts;
```

```
                }

                public static class SpeechVoice implements IProguardKeeper {
                        //Name of voice source
                        public String name = "";
                        //Gender of voice source - 0: male; 1: female
                        public int sex;
                        //Whether voice source is an adult - 0: minor; 1: adult
                        public int adult;
                }
        }
```

Note: The speech engine includes speech recognition and TTS functions.

3) Speech engine application replacement

Meta-data configuration

```
<meta-data    android:name="cruiser_speech" android:value="ubt" />
```

Note: The "value" must be the **engine of VoiceEngine** in the above config, and is "ubt" by default here. Third-party speech engines use custom strings.

```
<service
        android:name="com.ubtechinc.transportlib.messager.MessageService"
        android:enabled="true"
        android:exported="true">
        <intent-filter>
            <action android:name="com.ubtechinc.speechservice.demo.speech_service" />
        </intent-filter>
</service>
```

Note: The action name of MessageService is the application's **applicationId+".speech_service"**
Note: IPC MessageService configuration (MessageService is provided by the SDK)

```
mSpeechMessager = SpeechMessager.createMessager(new SpeechListener() {

        @Override
        public void build(String content) {
            SpeechLog.i("buildGrammar content:" + content);
            if (mSpeechRecognizer != null) {
                mSpeechRecognizer.buildGrammar(content);
            }
        }

        @Override
        public void startRecognize() {
            SpeechLog.i( "startRecognize");
            if (mSpeechRecognizer != null) {
                mSpeechRecognizer.startRecognize();
            }
        }

        @Override
        public void stopRecognize() {
            SpeechLog.i( "stopRecognize");
            if (mSpeechRecognizer != null) {
                mSpeechRecognizer.stopRecognize();
            }
        }

        @Override
        public void startSpeaking(String txt) {
            SpeechLog.i( "startSpeaking");
            if (mSpeechTTs != null) {
                mSpeechTTs.startSpeaking(txt);
            }
        }

        @Override
        public void stopSpeaking() {
            SpeechLog.i( "stopSpeaking");
            if (mSpeechTTs != null) {
                mSpeechTTs.stopSpeaking();
            }
        }
        @Override
        public void startListening() {
            SpeechLog.i( "startListening");
            if (mSpeechIat != null) {
                mSpeechIat.startListening();
            }
        }
        @Override
        public void startListening(int time, int eostime) {
            SpeechLog.i( "startListening");
            if (mSpeechIat != null) {
                mSpeechIat.startListening(time,eostime);
            }
        }

        @Override
        public void stopListening() {
            SpeechLog.i( "stopListening");
            if (mSpeechIat != null) {
                mSpeechIat.stopListening();
            }
        }
```

```
        @Override
        public void setVoiceName(String name) {
            SpeechLog.i( "setVoiceName:"+name);
            if (mSpeechTTs != null) {
                mSpeechTTs.setVoiceName(name);
            }
        }

        @Override
        public void setTtsSpeed(String speed) {
            SpeechLog.i( "setTtsSpeed:"+speed);
            if (mSpeechTTs != null) {
                mSpeechTTs.setTtsSpeed(speed);
            }
        }

        @Override
        public void setTtsVolume(String volume) {
            SpeechLog.i( "setTtsVolume:"+volume);
            if (mSpeechTTs != null) {
                mSpeechTTs.setTtsVolume(volume);
            }
        }
    });
```

Note: SpeechMessager.class** provides IPC message sending and receiving functions. Third-party developers need createMessager to build the SpeechListener interface and then connect to the system voice control. For detailed usage examples, see the demo.

4) Semantic engine application replacement

Meta-data configuration
<meta-data android:name=" cruiser_nlp" android:value="ubt" />
    The "value" must be the **engine of NLP Engine** in the above config, and is "ubt" by default here.
    Third-party semantic engines use custom strings.

```
 <service
        android:name="com.ubtechinc.transportlib.messager.MessageService"
        android:enabled="true"
        android:exported="true">
            <intent-filter>
                <action android:name="com.ubtechinc.nlpservice.demo.nlp_service" />
            </intent-filter>
</service>

mIMessageSender = NlpMessager.createMessager(new NlpListener() {
            @Override
            public void onNlp(String request, String uuid) {
                handleNlp(request, uuid);
            }
        });
```

3.3 State Machine Integration

To make it easier to control the states centrally, the system now uses State Machine 2.0 to manage relevant state logics. Normally, only system built-in applications can connect to the state machine, while third-party developers will not be assigned to states and can only listen to relevant state changes. This is called from the launching Application of the app. You can use speech-related interfaces only after a successful callback is received. Here is the example code:

```
SystemStatusApi.get().registerStatusCallback(new RemoteStatusListener() {
                    @Override
                    public void onResult(List<Integer> stack, boolean change, int status) {
                         //This callback will be triggered when a system state machine update is available. The "list"
contains the current state id value.
                    }

                    @Override
                    public void onStatusPause(int paused, int pausedBy, String sponsor) {
                         //The current application's state is forced offline by another state
                    }

                    @Override
                    public void onStatusFree(final int laststatus) {
                         //Callback of system state machine in the idle state
                    }
                });
```

3.4 Navigation Control Integration

The system allows third-party developers to integrate the SDK to enable functions including map setup, obtaining and using a map, starting positioning, canceling positioning, searching for map navigation points, and navigation. Refer to the demo for the method of use. Here are relevant instructions and interfaces:

3.4.1 Initialization

Add the method NavigationApi.get().initializ(this) to onCreate() in the Application class of the app to initialize the navigation module.

3.4.2 Registering event callback

Add the following code where the app needs to listen to navigation event callback. See the demo example for details.

```
RosRobotApi.get().registerCommonCallback(RemoteCommonListener listener);
```

Note: In the demo, registerCommonCallback registers the event callback for map setup, starting positioning, and canceling positioning. For more information about RemoteCommonListener, refer to the definition of ROS interface calling listener interface in this document.

```
NavigationApi.get().setNavigationApiCallBackListener(NavigationApiCallBackListener navigationApiCallBackListener);
```

Note: In the demo, setNavigationApiCallBackListener registers the event callback for listening to navigation functions.

NavigationApiCallBackListener is defined as follows. For detailed usage, please refer to the demo example:

```
public interface NavigationApiCallBackListener {

   void onNavigationResult(int status, float x, float y, float theta);

   void onRemoteCommonResult(String pointName, int status, String message);
}
```

3.4.3 Obtaining the current map

The SDK interface allows developers to obtain the name of the map being used. Developers can call the ROS interface to obtain the path of the map currently being loaded. Here are the interface details:

```
String rosMapPath = RosRobotApi.get().getCurrentMap();
```

As the return value is the map path, to get the map name, you need to have the application itself extract the file name. Here is a demo code:

```
private String getCurMapName() {
            String curMapName = "";
            try {
                  //Call the ROS interface to obtain the path of the map currently being loaded
                  String rosMapPath = RosRobotApi.get().getCurrentMap();

                  if (StringUtils.isEmpty(rosMapPath)) {
                        return "";
                  }

                  //Obtain the map name from the ROS map path
                  curMapName = getFileName(rosMapPath);

            } catch (Exception e) {
                  e.printStackTrace();
                  curMapName = "";
            }
            return curMapName;
      }

private String getFileName(String filePath) {
      String fileName = "";
```

```
        if (StringUtils.isEmpty(filePath)) {
                return filePath;
        }

        int fp = filePath.lastIndexOf(File.separator);

        if (fp == -1) {
                return filePath;
        }

        fileName = filePath.substring(fp + 1);

        fp = fileName.indexOf("?");

        if (fp == -1) {
                return fileName;
        }

        fileName.substring(0, fp);

        return fileName;
}
```

3.4.4 Setting up the current map

The SDK provides the interface for setting up the map to be used. Developers can set the map to be loaded by syncing the map name to the navigation module through Cruzr PC software. See the demo example for details. Here is the main code:

```
Int mSetCurMapSectionId = RosRobotApi.get().setCurrentMap(String mapName);

public void onResult(int sectionId, int status, String message) {
      if(mSetCurMapSectionId == sectionId){

        switch (status) {
          case RosConstant.Action.ACTION_FINISHED:
          case RosConstant.Action.ACTION_CANCEL:
          case RosConstant.Action.ACTION_BE_IMPEDED:
          case RosConstant.Action.ACTION_FAILED:
          case RosConstant.Action.ACTION_DEVICE_CONFLICT:
          case RosConstant.Action.ACTION_EMERGENCY_STOPPED:
          case RosConstant.Action.ACTION_ACCESS_FORBIDDEN:
          case RosConstant.Action.ACTION_UNIMPLEMENTED:
            break;
          default:
            break;
        }
```

```
            return;
        }
}
```

Note: When a map is set, the SectionId of the corresponding operation will be returned. When registering the listening for RemoteCommonListener, the operation result of the corresponding SectionId will be returned. The application can then process the execution result according to the operation result status. For status definition, please refer to the definition of returned parameters in RemoteCommonListener of SDK.

3.4.5 Starting map positioning

The SDK provides the map positioning function for the robot. Every time a map is set to be used, the robot will clear its current location and will need to re-position. The robot can only use the navigation function and get the correct location when obtaining the current location after positioning. If a robot has not completed positioning, it will fail when using the navigation function, and a null default value will be returned when it obtains the current location. The SDK provides 2 positioning interfaces. One is global positioning, where the robot searches matching results on the entire map, and the possibility of successful positioning is low. The other one is local positioning, where the robot searches matching results within 1.5 m to 3 m around the possible location coordinates that have been passed to the robot. See the demo example for details.

```
Int mStartLocationSectionId = RosRobotApi.get().navigateRelocationCtrl(0);
```

Note: Global positioning. Passing parameters: 0 - starting global positioning; 1- canceling positioning.

```
//Local positioning
MapPointModel mapPointModel = mapPointModels.get(0);
Float mapX = Float.valueOf(mapPointModel.getMapX());
Float mapY = Float.valueOf(mapPointModel.getMapY());
Float mapTheta = Float.valueOf(mapPointModel.getTheta());
mStartLocationSectionId = RosRobotApi.get().navigateRelocationStartByPos(mapX,mapY,mapTheta);
```

When positioning is started, the SectionId of the corresponding operation will be returned. When registering the listening for RemoteCommonListener, the operation result of the corresponding SectionId will be returned. The application can then process the execution result according to the operation result status. For status definition, please refer to the definition of returned parameters in RemoteCommonListener of SDK.

3.4.6 Canceling map positioning

The SDK provides the function of canceling positioning. This uses the same interface as starting global positioning, but passes a different parameter. See the demo example for details. Here is the main code:

```
Int mStopLocationSectionId = RosRobotApi.get().navigateRelocationCtrl(1);
```

Note: Cancel positioning. Passing parameter: 1- canceling positioning.

When positioning is canceled, the SectionId of the corresponding operation will be returned. When registering the listening for RemoteCommonListener, the operation result of the corresponding SectionId will be returned. The application can then process the execution result according to the operation result status. For status definition, please refer to the definition of returned parameters in RemoteCommonListener of SDK.

3.4.7 Querying all map points

The SDK provides the interface for querying all navigation location points on the map. Developers can edit the navigation points in the Cruzr PC software, and then sync them to the robot. Cruzr can call the NavigationApi interface to navigate to the location point. See the demo example for details. Here is the main code for querying location points:

```
List<MapPointModel>    mapPointModels = NavigationApi.get().queryAllMapPointByMapName(mapName);
```

MapPointModel is defined as follows:

```
public class MapPointModel {
        public enum EnumPointType {
                /**
                  * Standard location
                  */
                normal_position,
                /**
                  * Tour location
                  */
                travel_position,
                /**
                  * Charging location
                  */
                charge_position,
        }

        /**Unique database ID of point*/
        private int id;
        /**Name of point*/
        private String pointName;
        /**Unique database ID of map*/
        private int mapId;
        /**Name of map*/
        private String mapName;
        /**Point type: normal, patrol, charging*/
        private String pointType;
        /**Position on the x axis of the map file, represented by meters*/
        private String mapX = "0";
        /**Position on the y axis of the map file, represented by meters*/
        private String mapY = "0";
        /**Theta on the z axis of the map file, represented by radians from –π to π*/
        private String theta = "0";
        /**Position on the x axis of the displayed image, represented by pixels*/
        private String displayX;
        /**Position on the y axis of the displayed image, represented by pixels*/
        private String displayY;
        /**Point description*/
        private String description;
        /**Whether set to standby location point*/
        private boolean isStandby;
```

```
    /**Whether set to greeting location point*/
    private boolean isWelcome;
    /**Whether set to patrol location point*/
    private boolean isCruiser;
    /**Sequence number of patrol location points. The default value is –1, and it can be set starting from 0.*/
    private int cruiserIndex = -1;
}
```

3.4.8 Starting and canceling navigation

The SDK provides the interface for navigation so that the robot can navigate to the destination through the NavigationApi interface. Here is the interface for starting navigation. Please see the demo example:

```
public void startNavigationService(String pointName)

public void stopNavigationService()
```

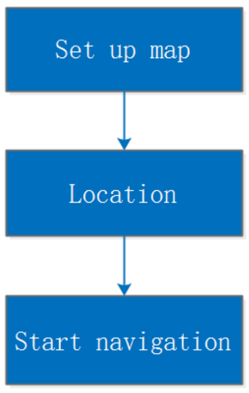
When navigation is started, there will be a callback of the corresponding result. Registering the listening of NavigationApiCallBackListener will call back the listening of NavigationApiCallBackListener. The application can process the execution result of the callback method. For the callback method and parameter definition of the NavigationApiCallBackListener listening, please refer to 3.4.2 Registering Event Callback.

3.4.9 Obtaining the current location

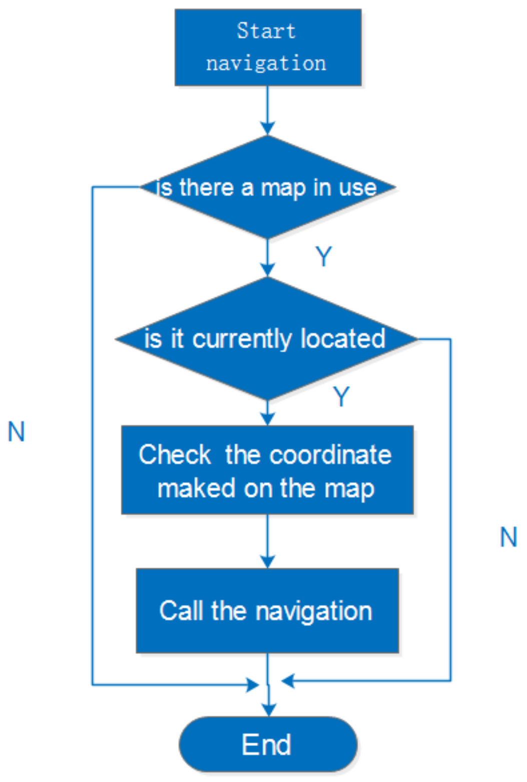
The SDK provides the interface for obtaining the current location. The current location interface returns the coordinates of the robot's current location. If the robot has not completed positioning, a default null value will be returned. The unit of the returned coordinates is meters. To convert it to the xy coordinates that show the map image, you need to combine the MapModel scale and coordinate system. See the demo example for code details. Here is the main code:

```
public static final float com.ubtechinc.cruzr.sdk.navigation.POSITION_INVALID_VALUE = 0x7f800000;

//Obtain the current coordinates (ROS coordinate system) of the robot represented by meters
Position curPosition = RosRobotApi.get().getPosition(false);

public MapModel queryMapModelByMapName(String mapName)

public float[] convertMapPositionToDisplayPosition(MapModel mapModel, Position position) {
    if (null == mapModel || null == position) {
        return null;
    }

    float resolution = Float.valueOf(mapModel.getResolution());
    float mapLeftTopX = Float.valueOf(mapModel.getBmp_x());
    float mapLeftTopY = Float.valueOf(mapModel.getBmp_y());

    float x = -mapLeftTopX + position.x / resolution;
    float y = mapLeftTopY - position.y / resolution;
    float theta = (float) (180 * position.theta / Math.PI);
```

```
        return new float[]{x, y, theta};
}
```



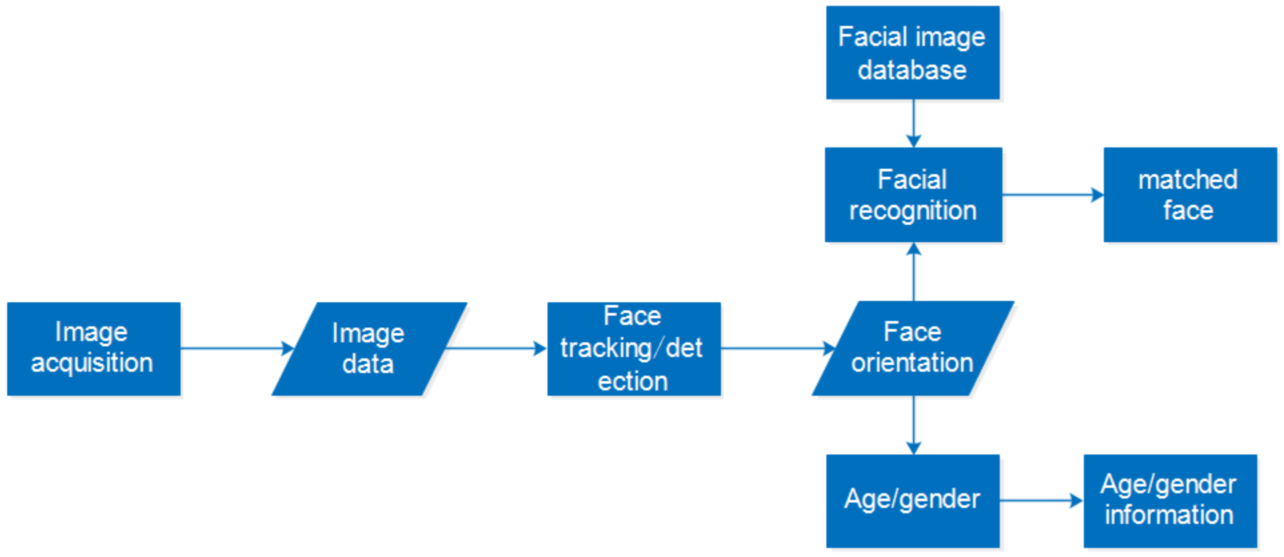
Using map workflow



Using navigation workflow

3.5 Facial Recognition Integration

The system allows third-party developers to integrate the SDK and enable functions including face tracking, face detection, and facial recognition. Developers need to save and load the image collection and face database data by themselves. See the documentation for detailed entity definitions and interface instructions, and refer to the demo for the method of use. Here are relevant instructions and interfaces:



3.5.1 Initialization

The reference code is as follows:

```
ASFREngine tracker = new ASFREngine();
tracker.open();
//Obtain raw image data
RawImage image;
ByteBuffer mPreviewBuffer;
int mPreviewWidth;
int mPreviewHeight;

//Set bufferSize to the length of preview frame data according to preset camera preview parameters
private void init(int bufferSize){
        mPreviewBuffer = ByteBuffer.allocateDirect(bufferSize);
        mPreviewBuffer.order(java.nio.ByteOrder.LITTLE_ENDIAN);
        mPreviewBuffer.position(0);
        image = new RawImage();
}

//Register callback and obtain camera preview frame data
@Override
public void onPreviewFrame(byte[] data, Camera camera) {
        mPreviewBuffer.put(data);
        image.setData(mPreviewBuffer);
        image.setWidth(mPreviewWidth);
        image.setHeight(mPreviewHeight);
        image.setFormat(ArcSoftConstant.Format.ASVL_PAF_NV21);
}

//Start face detection
ASFaceDetection detection = tracker.detect(image);
```

```
//Close the engine
tracker.release();
```

3.5.2 Still image detection

The engine opens, starts detection, and closes in similar ways as above. Here is how to use processed local image data as the engine input. The reference code is as follows:

```
//Path is the path of local images
private void init(String path){
    Bitmap bitmap = BitmapFactory.decodeFile(path);
    YUVFormat.NV21 nv21 = ImageUtils.getNV21ByBitmap(bitmap);
    image = new RawImage();
    image.setData(ByteUtils.wrapToByteBuffer(nv21.getData()));
    image.setFormat(ArcSoftConstant.Format.ASVL_PAF_NV21);
    image.setWidth(nv21.getWidth());
    image.setHeight(nv21.getHeight());
}
```

3.5.3 Image feature extraction

The engine opens, starts detection, and closes in similar ways as above. Here is how to call API for the recognition and matching processes. The reference code is as follows:

```
public FaceMatch recognize(ASFaceDetection detection, float similarity, List<FaceModel> source) {
    ASFREngine recognizer = new ASFREngine();
    recognizer.open();

    //1. Compute facial features, which takes 1000ms - 1350ms on Cruzr for a single person
    ASFaceRecognition recognition = recognizer.recognize(image, detection);

    //Failed to extract feature values
    if (recognition == null) {
        return null;
    }

    //2. Search the face database. The time consumed depends on the source size.
    FaceMatch faceMatch = recognizer.search(similarity, recognition, source);
    //Detect if it is interrupted
    if (recognizer.isInterrupted()) {
        return null;
    }

    if (faceMatch == null) {
        return null;
    }
}
```

3.6 Facial Expression Integration

Facial expression is a basic function that the system provides for third-party settings. Users can obtain the current list of expressions supported by the system through the API and use the rich array of expressions in different scenes. See the documentation for detailed interface instructions, and refer to the demo for the method of use. Here are the relevant interfaces:

```
//Call initialization in Application
CruzrFaceApi.initCruzrFace(this);

//Set a smiling expression, play music, and loop the expression
CruzrFaceApi.setCruzrFace(null,"face_smile",true,true);

//Obtain the ID of the currently displayed expression
String curFaceid = CruzrFaceApi.getCurrentFaceId(callback);

//Obtain the current list of available expressions and use the interface where you obtain the expression list to pass the callback object callBack. Then, return the result using the callback method onCruzrFaceListCallBack.
CruzrFaceApi.getCruzrFacesList(callback);

//Obtain the ID of the currently displayed expression
String curFaceid = CruzrFaceApi.getCurrentFaceId(callback);

//Obtain the expression name according to the expression ID. callBack can be null, and the expression name will be directly returned. In simplified Chinese, the following code will return "开心".
String faceName = CruzrFaceApi.getCruzrFaceNameByFaceId(callback,"face_happy");
```

3.7 Dance Integration

Dance is the function the system provides for third parties to integrate dances. There are 5 dances built into the system currently. To ensure program scalability, developers should obtain the names of currently supported dances in a dynamic way. See the documentation for detailed interface instructions, and refer to the demo for the method of use. Here are the relevant interfaces:

```
List<String>danceNameList = DanceControlApi.getInstance().getDanceList();

booleanisDancing = DanceControlApi.getInstance().isDancing();

String dance = DanceControlApi.getInstance().getCurrentDance();

DanceControlApi.getInstance().initialize(this, new DanceConnectionListener() {
@Override
public void onConnected() {
    //TODO is successfully connected to the dance service
}
@Override
public void onDisconnected() {
      //TODO is disconnected from the dance service
}

@Override
public void onReconnected() {
      //TODO is reconnected to the dance service
}
});
```

Note: When calling the initialize method, you can pass the connecting listener to listen to the current connection status in real time. The other calling method of DanceControlApi will be effective only when a connection is established.

```
DanceControlApi.getInstance().dance(danceNameList.get(0), new RemoteDanceListener() {
@Override
public void onResult(intstatus) {

      switch (status){
            case DanceConstant.STATE_DANCE_START:
                  //TODO Dance starts
                  break;
             case DanceConstant.STATE_DANCE_COMPLETE:
                  //TODO Dance completes
                  break;
             case DanceConstant.STATE_DANCE_CANCEL:
                  //TODO Dance is canceled
                  break;
             case DanceConstant.STATE_DANCE_FORBIDDEN:
                  //TODO Robot arms are locked. Dancing is not allowed.
                  break;
             case DanceConstant.STATE_DANCE_FAIL:
                  //TODO Dance failed
                  break;
            }
    }
});
```

Note: When starting a dance, pass in the dance name to request performance of the corresponding dance. RemoteDanceListener listens to the performance status of this dance task.

```
DanceControlApi.getInstance().stop();
```

Note: When performing a dance, you can call stop to cancel the task. At the same time, the listener registered when starting this dance will receive a notification that the task is canceled (DanceConstant.STATE_DANCE_CANCEL).

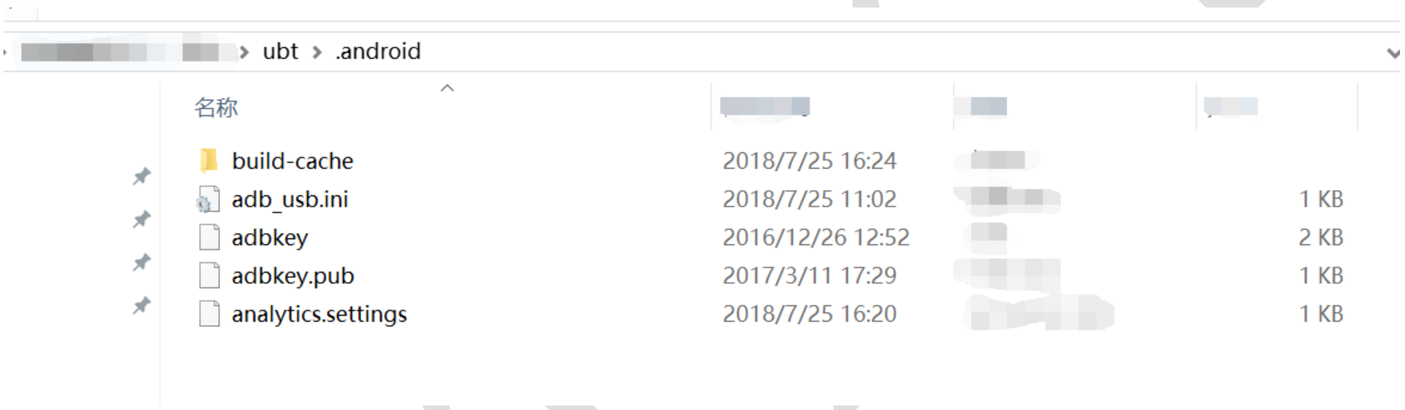
```
DanceControlApi.getInstance().registerDanceServiceListener(new DanceServiceListener() {
      @Override
```

```
            public void onChange(String name, intstatus) {
                //TODO Listen to the dance name and status of the dance service being performed
            }
        });
```

DanceControlApi.getInstance().unregisterDanceServiceListener();
Note: Register the listener for the dance service status to listen to the performance status of each dance task. When listening is no longer needed, call the unregistering interface to stop listening.

4. ADB Debugging

The ADB debugging of Cruzr robots requires dedicated ADB files.
Copy the dedicated adbkey and adbkey.pub files to the corresponding path as shown below and override the original files, and then restart Android Studio.
Now you can see the log output.



› ubt › .android

名称			
build-cache	2018/7/25 16:24		
adb_usb.ini	2018/7/25 11:02		1 KB
adbkey	2016/12/26 12:52		2 KB
adbkey.pub	2017/3/11 17:29		1 KB
analytics.settings	2018/7/25 16:20		1 KB

5. FAQs

1. How can I show and hide the navigation bar?

A: You can show and hide the navigation bar through broadcasting as follows:
public static String SHOW_BAR = "com.ubt.cruzr.showbar";
public static String HIDE_BAR = "com.ubt.cruzr.hidebar";

2. Does voice wakeup notify third parties?

A: Voice wakeup, virtual button wakeup, and vision wakeup all broadcast to third parties. Please listen to the following broadcasting:
public static final String CORE_SERVICE_WAKE_UP = "com.ubtechinc.cruzr.coreservices.ACTION.WAKE_UP";
public static final String CRUISER_DATA_WAKE_UP_TYPE = "wakeup_type";
public static final int WAKE_UP_TYPE_SPEECH = 0;
public static final int WAKE_UP_TYPE_KEY = 1;
public static final int WAKE_UP_TYPE_FACE_IN = 2;

3. How can I disable voice wakeup?

The system voice provides relevant interfaces to control voice, button, and vision wakeup. Once set to false, the relevant wakeup function will be disabled and the system will not record nor perform recognition. If set to true, the function will work normally again, and the wakeup broadcasting will not be sent. Third-party developers can use this method if they want to integrate their own speech engine, as well as make their own recordings for wakeup recognition and speech recognition. The reference code is as follows:

```
//Disable voice wakeup
SpeechRobotApi.get().enableWakeup(SpeechConstant.PROHIBIT_WAKE_UP_TYPE_SPEECH,false);
//Disable button wakeup
SpeechRobotApi.get().enableWakeup(SpeechConstant.PROHIBIT_WAKE_UP_TYPE_KEY,false);
//Disable vision wakeup
SpeechRobotApi.get().enableWakeup(SpeechConstant.PROHIBIT_WAKE_UP_TYPE_FACE,false);
//Disable all wakeup
SpeechRobotApi.get().enableWakeup(SpeechConstant.PROHIBIT_WAKE_UP_TYPE_ALL,false);
```

4. How do third parties integrate their own NLP?

The system allows third parties to use the built-in speech engine and their own semantic engine. Developers can choose to receive all system voice commands or only online commands by setting up interfaces. To do this, developers must integrate the system voice command issuing mechanism first. For details about integrating the voice command issuing mechanism, please refer to 3.2.1 Initialization. Here is the code:

```
//Issue system voice commands normally
SpeechRobotApi.get().speechPermissionDispatch(103,SpeechConstant.SPEECH_DISPATCH_PERMISSION_NONE);
//Issue all online commands except the system built-in offline commands to the specified third-party app
SpeechRobotApi.get().speechPermissionDispatch(103,SpeechConstant.SPEECH_DISPATCH_PERMISSION_NLP);
//Issue all system voice commands to the specified third-party app
SpeechRobotApi.get().speechPermissionDispatch(103,SpeechConstant.SPEECH_DISPATCH_PERMISSION_ALL);
```

Note: The 103 in the example is the appid for the current developer

5. Is there a specific interface for detecting objects?

A: Yes. The system supports laser radar detection of objects in a range of 1.3 m, and enables recognition by default when it is started. Developers listen to and report events through registerWarnCallback (used for notifications about system alerts, battery, and sensors) and distinguish them using "key". For example, 210 belongs to near field detection, and reporting continues after an object is detected. Before using this API, you must complete initialization first. The reference code is as follows:

```
private static final int KEY_DETECT = 210;
private static final int DETECT_OBJECT_IN = 1;
private static final int DETECT_OBJECT_OUT = 2;

RosRobotApi.get().registerWarnCallback(new RemoteWarnListener() {
                @Override
                public void onResult(String message, int key, int value, int data) {
                    if (key == KEY_DETECT) {
                        switch (value) {
                            case DETECT_OBJECT_IN:
                                //Object comes
                                break;
                            case DETECT_OBJECT_OUT:
                                //Object leaves
                                break;
                        }
                    }
                }
            });
```

6. How does an action interface listen to the status callback?

A: Generally, a unique sectionId will be returned every time an interface is called. The parameter sectionId in the asynchronous callback indicates the status callback of the current calling.
The parameter sectionId in the asynchronous callback will correspond to this. The callback information is the content corresponding to the current sectionId. Since now the listening is based on the last callback, developers are advised to use global listening and determine the current calling by sectionId. Currently, listening related to the RosRobotApi callback mechanism are all processed in this way.

```
int sectionId = RosRobotApi.get().run("hug", mRemoteCommonListener);

private RemoteCommonListener mRemoteCommonListener = new RemoteCommonListener() {
        @Override
        public void onResult(int sectionId, int status, String message) {
            //Status is the callback of the execution status of the relevant action, where 2 indicates execution is completed. For a detailed definition, refer to the action status definition in RosConstant.Action.
        }
    };
```

7. Can I listen to the state machine changes?

A: Yes. Currently the state machine only assigns a state to the system and meets the requirement of system interaction. Third-party apps are not recommended to integrate the state machine. If you need to listen to status changes, refer to the following code (see SystemStatus definition in the documentation for status values):

```
SystemStatusApi.get().registerStatusCallback(new RemoteStatusListener() {
                    @Override
                    public void onResult(List<Integer> stack, boolean change, int status) {
                      //The current state machine changes -> status;
                    }

                    @Override
                    public void onStatusPause(int paused, int pausedBy, String sponsor) {
                    }
                    @Override
                    public void onStatusFree(final int laststatus) {
                        //Idle callback of the current state machine. The last status is laststatus.
                    }
                });
            }
        });
```

8、When obtaining the current location during navigation, why is 0x7f800000 returned?

A: As the robot has not completed positioning, the default null value 0x7f800000 is returned when obtaining the current location. See 3.4.9 Obtaining the current location.

9、Why can't I navigate?

A: To navigate, the following conditions must be met: 1. the map to be used has been set up; 2. the robot has completed positioning; 3. the navigation destination exists.

10、Why can't I set up a map successfully? Why haven't I received a success notification?

A: If map setup fails, please use Cruzr PC software to re-sync the map. Go to Settings > Movement Settings and see if the chassis function is disabled. Navigation will fail when the chassis function is disabled. If it still does not work, you need to turn on backdoor diagnostics to see if there is an exception in the SLAM navigation module.

11、How can I make the robot set up a map and complete positioning automatically after startup and then navigate to a destination?

A: Receive the following broadcasting of successful connection between the main service and ROS after startup to set the map to be used. If map setup fails, try again after a couple of seconds. Then, in the callback of a successful map setup, execute the positioning function through global positioning or by passing the robot's possible coordinates. Start navigation and listen to the callback of the positioning result. After successful positioning, the navigation function can be started.

```
/**Establish broadcasting between the main service and ROS*/
public static final String SDK_READY_ACTION = "com.ubtechinc.cruzr.coreservices.ACTION.WORK_ON";
```

12、How can I make the robot raise its head?

A: Our API supports head control (raising and lowering the head). Users can customize the parameters to control the speed of head raising. For detailed usage, refer to the following code:

```
//Head 180-270 degrees
public static final String HEAD = "HeadPitch";
public static final double MIN_HEAD_ANGLE = 180 * M_PI / 180;
public static final double MAX_HEAD_ANGLE = 270 * M_PI / 180;
public static final double PER_SECOND_ANGLE = 30 * M_PI / 180;
//Angle adjustment threshold
public static final double ANGLE_ADJUST_THRESHOLD = 0.1;

public static double calculateMoveTime(double angle) {
    double duration = Math.abs(angle) / PER_SECOND_ANGLE;
    if (duration < 1) {
        duration = 1;
    }
    return duration;
}

public static void adjustSightTo(double angle) {
        double currentAngle = RosRobotApi.get().getJointCurrentAngle(HEAD);
        double difference = angle - currentAngle;
        if (Math.abs(difference) < ANGLE_ADJUST_THRESHOLD) {
                return;
        }
        float duration = (float) calculateMoveTime(difference);
        RosRobotApi.get().setAngles(HEAD, (float) angle, duration);
}

adjustSightTo(MAX_HEAD_ANGLE );
```