一、版本历史

版本号	修改内容简介	修改日期	修改人
V1.0	创建文档	2021-08-23	wangchangming
V1.1	更新文档: * 更新连接方式，默认初始化AOA, 参见第四章; * 文档新增ServiceTypes常量定义； * 支持渠道号初始化时配置，参见4.2小节; * 车厂接入时，协议版本号需设置为4. * 更新8.3小节，新增模块状态控制处理.	2021-11-15	wangchangming
V1.2	更新文档: * 新增连接主动断开接口； * 阐述setFeatures和setFeature的区别； * 新增蓝牙音频描述。	2021-11-29	wangchangming
V1.3	更新文档: * 录音相关代码从SDK中挪到Demo中，更新了MIC收音相关文档;	2021-12-13	wangchangming
V1.4	更新文档: * 新增无感连接注意事项	2021-12-21	wangchangming
V1.5	更新文档: * 新增aac音乐编码压缩； * 第四章节连接新增超时断开设置为8秒说明。	2022-02-21	wangchangming chenyanqiang

二、工程配置

2.1 SDK集成

- 集成aar

需要将carlife-sdk-release文件复制到Application Module/libs文件夹（没有的需手动创建），并将以下代码添加到您app的build.gradle中:

```
repositories {
    flatDir {
        dirs 'libs'
    }
}
depedencies {
    compile(name: 'carlife-sdk-release', ext: 'aar')
}
```

- 添加依赖

在主Module的build.gradle文件添加sdk的依赖

```
dependencies {
    implementation 'com.google.protobuf:protobuf-lite:3.0.1'
}
```

- AndroidManifest配置

```
<!--必要权限-->
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

- 运行环境配置

本SDK需运行在Android5.0(API Level 21)及以上版本。

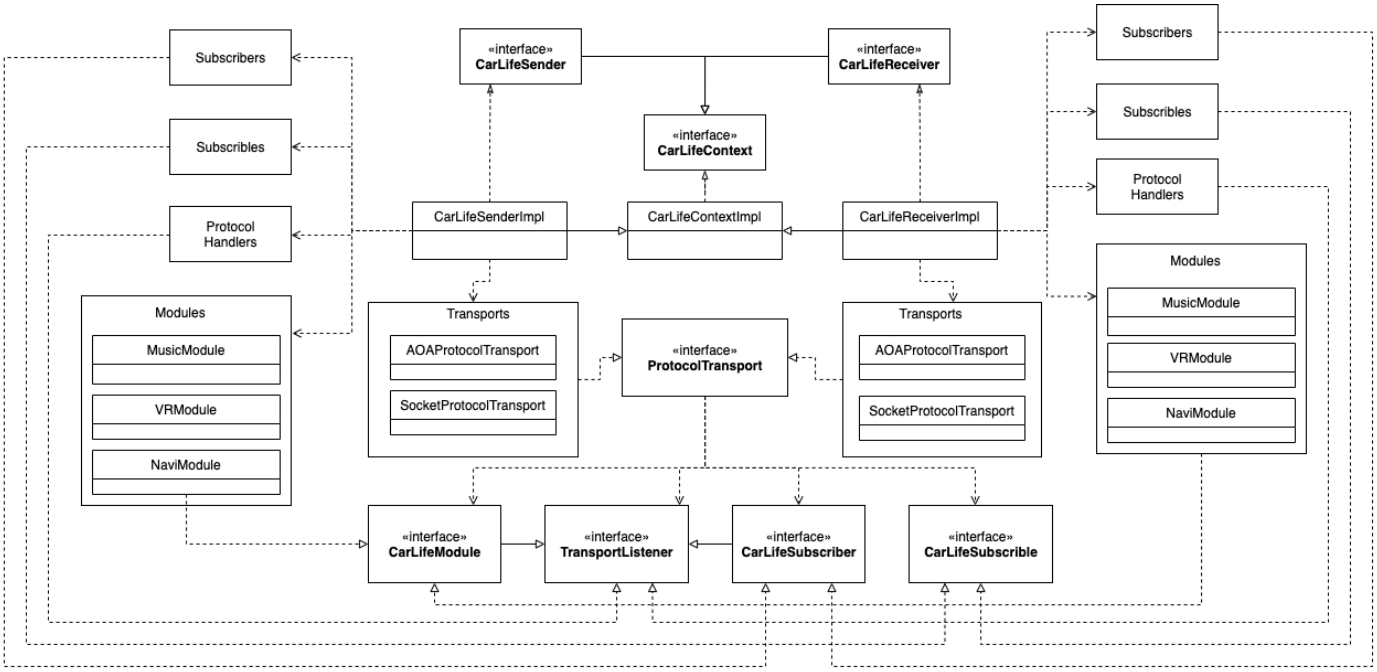
- 代码混淆

如果你需要使用proguard混淆代码，需确保不要混淆SDK的代码。请在proguard-rules.pro文件(或其他混淆文件)尾部添加如下配置。

```
-keep class com.baidu.carlife.** {*;}
-dontwarn com.baidu.carlife.**
```

2.2 SDK架构

2.2.1 类结构



SDK的里面封装了车机端及手机端相关的连接投屏相关流程，以及消息订阅，模块管理等内容，减少了车厂接入车机端开发时的工作量。其中的CarLifeReceiver代表车机端，其实现者为CarLifeReceiverImpl，封装了订阅、模块管理、连接投屏相关的操作流程。

2.2.2 接口说明

这里将简单介绍车机端开发时涉及到的重要接口，有的SDK已经实现，有的还需要车机端自己拓展功能，下面将简单介绍一下相关概念，如需车厂实现部分将在后面部分进行详解，这里仅简单了解相关概念。

- CarLifeContext

CarLife上下文，通用方法封装，通过CarLifeContext可以发送消息、监听状态、加解密等等。

- CarLifeModule

CarLife本身具有模块的概念，并定义了导航、音乐、语音等几个模块，模块本身具备id（唯一表示）、state（状态）。当状态发生变化时，要实时同步给车机。CarLifeContext内部封装了具体实现，使模块实现者只关注业务实现，无需关心协议交互。

- CarLifeSubscriber及CarLifeSubscrible

CarLife中有消息订阅机制，即车机可以跟手机订阅消息（路口信息、红绿灯信息等），手机也可以跟车机订阅消息（GPS、车速等）有关的协议交互来完成。CarLifeContext封装了相关协议交互，对外只暴露接口。

- CarLifeReceiver

CarLifeReceiver车机端抽象接口，具备车机端的特有能力，反控事件的发送、语音收音、音频播放与焦点管理等。

- CarLifeSender

CarLifeSender手机端抽象接口，具备手机端的特有能力，虚拟屏幕创建、反控事件处理、硬按键处理、文件传输相关。

车厂接入时无需关心CarLifeSender。，此仅与手机端有关.

- ProtocolTransport

ProtocolTransport负责建立连接相关，包括AOA连接、WIFI连接、wifi p2p直连相关的连接，此类业务操作相关在SDK中完成，里面有完善的断开重连机制，也有连接过程中各个状态的回调，车机端连接只需调用CarLife.receiver().connect()即可。

2.3 设备要求

2.3.1 车机端最低硬件要求

Item	Requirement
Physical Connection	USB 2.0 HiSpeed Host: * Standard USBA port; * 480 Mbits/s throughput; * 4.75~5.25V @ 500mA power to MD. Bluetooth: * HandsFree Profile 1.5; * Secure Simple Pairing ; WiFi: * WiFi-Direct; * 5.0G Hz.
High Resolution Display	Minimum resolution: 800x480 pixels(recommend) Close to 1:1 pixel aspect ratio (square pixels)
Video	H.264 hardware Decoding: * H.264 Base Line Profile level 3.1 is mandatory (Both sides to guarantee 800x480, 60fps; For 720p 30fps)
Audio	Supported Formats: * PCM. Audio data Streams: * Voice stream(Mobile Device->HU)&&Navigation TTS; * Media stream; * Bluetooth in-call audio; * VR (HU->Mobile Device).
User Input Devices	Below physical or soft buttons are supported: * Touch screen; * Knobs, buttons; * Physical or soft “Home” button to switch Car APP and Native APPs.
Speakers	The car must provide: * Audio output via speakers.
MFI	MFI certification chip USB OTG switch: * Usb host mode;

	* Usb device mode.
Car Sensors(optional)	Below car data is required: * Car speed; * Gro; * Accelerometer; * GPS.
MIC(optional)	Below are mandatory for VR function: * Voice PCM flow; * 16bit ; * amplitude: about 5000; * SNR > 12dB.

三、SDK初始化配置

3.1 初始化配置

> 重点：
>
> 1. CarLife SDK 需要在主线程中初始化；
> 2. 初始化时需要传入相关配置。

开发者需要在Application#onCreate()方法中调用以下代码来初始化CarLife SDK 。

```
class VehicleApplication : Application() {

    var vehicleBind: VehicleService.VehicleBind? = null

    override fun onCreate() {
        super.onCreate()
        CarlifeConfUtil.getInstance().init()
        initReceiver()
    }

    @SuppressLint("ClickableViewAccessibility")
    private fun initReceiver() {
        val screenWidth = resources.displayMetrics.widthPixels
        val screenHeight = resources.displayMetrics.heightPixels
        val displaySpec = DisplaySpec(
            this,
            screenWidth.coerceAtLeast(screenHeight).coerceAtMost(1920),
            screenWidth.coerceAtMost(screenHeight).coerceAtMost(1080),
            30
        )

        val features = mapOf(
```

```
            FEATURE_CONFIG_USB_MTU to 16 * 1024,
            FEATURE_CONFIG_FOCUS_UI to 1,
            FEATURE_CONFIG_I_FRAME_INTERVAL to 300
        )

        val configs = mapOf(
            CONFIG_LOG_LEVEL to Log.DEBUG,
            CONFIG_SAVE_AUDIO_FILE to true,
            CONFIG_WIFI_DIRECT_NAME to "MI8",
            CONFIG_CONTENT_ENCRYPTION to true,
            CONFIG_USE_ASYNC_USB_MODE to false,
            CONFIG_PROTOCOL_VERSION to 3,
            CONFIG_CONNECT_TYPE to 1,
            CONFIG_HU_BT_NAME to "hu_name",
            CONFIG_HU_BT_MAC to "xxx-xxx-xxx"
        )

        Log.d(Constants.TAG, "VehicleApplication initReceiver $screenWidth,
$screenHeight $displaySpec")
        CarLife.init(
            this,
            "20029999",
            "12345678",
            features,
            CarlifeActivity::class.java,
            configs)
        VoiceManager.init(CarLife.receiver())
        CarLife.receiver().setDisplaySpec(displaySpec)
    }
}
```

3.2 CarLife初始化接口说明

```
CarLife.init(
            this,
            "20029999",
            "12345678",
            features,
            CarlifeActivity::class.java,
            configs)
```

CarLife类里面有两个静态方法，init和receiver()，还有一个私有成员变量receiver。其中init方法用于初始化receiver的，此方法必须在主线程中调用；receiver是用于返回receiver的值。这里初始化完成后，**CarLife.receiver()** 的返回就不为空，后续就可以使用了。

3.2.1 方法说明

方法名	方法介绍
init(Context context, String channel, String cuid, Map<String, Int> features, Class activityClass, Map<String, Any> configs)	用于初始化CarLife SDK，无返回值
receiver()	返回CarLife SDK的上下文对象

3.2.2 参数说明

init方法

参数说明

参数	含义
context	Context: Application上下文对象
channel	String: 车机渠道号，用于手机端验证车机端的合法性
cuid	String: 车机端设备唯一标识符
features	Map<String, Int>: 车机端支持的功能集合。（如车机端是否支持语音唤醒） 一般设置一些固定的feature，无需从配置文件读取的。从配置文件读取的另有API接口设置
activityClass	Class: 车机端App的主Activity
configs	Map<String, Any>: 车机端相关配置集合。（如车机端Wi-Fi直连的名称、是否加密等）

返回值

Returns	
Void	无返回值

receiver方法

参数说明

参数	含义
无参数	

返回值

Returns	
CarLifeReceiver	CarLife SDK的上下文对象，用于sdk的相关操作

3.2.3 features及configs说明

features及configs这里只是传入的一些配置，后面都可以使用**CarLife.receiver()** 调用相应方法进行修改。

features

Key	Value(Int)	含义
FEATURE_CONFIG_USB_MTU	16 * 1024	USB连接时，车机端指定的最大包的size。 注：可不指定，主要为了处理在某些车机端，包太大导致发送失败
FEATURE_CONFIG_FOCUS_UI	1	定制焦点态。1:可用 0: 不可用 默认不可用
FEATURE_CONFIG_I_FRAME_INTERVAL	300	关键帧间隔 默认为1
FEATURE_CONFIG_VOICE_MIC	1	语音MIC。 0表示使用车机MIC，1表示使用手机MIC，2表示车机端MIC不支持， 3表示车机端支持回声消噪，排列方式是先左声道\右声道， 4表示车机端支持回声消噪，排列方式是先右声道\左声道。 默认使用车机MIC
FEATURE_CONFIG_VOICE_WAKEUP	1	车机是否支持语音唤醒。0:不支持 1:支持 默认支持唤醒
FEATURE_CONFIG_BLUETOOTH_AUTO_PAIR	1	定制蓝牙自动匹配。0:不匹配 1:匹配 默认自动匹配
FEATURE_CONFIG_BLUETOOTH_INTERNAL_UI	0	定制蓝牙电话。0:不可用 1:可用 默认不可用
FEATURE_CONFIG_MEDIA_SAMPLE_RATE	0	媒体采样率，定制MD传入HU的PCM流的采样率 0:跟随手机系统 1:48k 默认不定制
FEATURE_CONFIG_AUDIO_TRANSMISSION_MODE	0	定制MD向HU传输音频流方式 0: 专用音频通道 1:蓝牙通道 默认为使用专用音频通道进行音频传输
FEATURE_CONFIG_ENGINE_TYPE	0	定制运行CarLife的车机类型 0:燃油车 1:电动车 默认为燃油车
FEATURE_CONFIG_INPUT_DISABLE	0	是否根据车速屏蔽输入法 0:是 1:否 默认是只要HU发过来车速过来就启用屏蔽

注：这里的功能定制最后会通过MSG_CMD_HU_FEATURE_CONFIG_RESPONSE协议传递给手机端。

configs

Key	Value(Object)	含义
CONFIG_LOG_LEVEL	Log.DEBUG	sdk里面的日志级别
CONFIG_SAVE_AUDIO_FILE	"MI8"	车机端wifi直连的名称
CONFIG_CONTENT_ENCRYPTION	true	是否加密
CONFIG_USE_ASYNC_USB_MODE	false	是否使用同步 默认为false，Android4.3及以下需设置为true
CONFIG_PROTOCOL_VERSION	3	车机端协议版本 手机端会验证此版本，当手机CarLife版本小于车机端版本时，会连接不成功
CONFIG_HU_BT_NAME	hu_name	车机端蓝牙名称
CONFIG_HU_BT_MAC	xxx-xxx-xxx	车机端蓝牙MAC地址
...		
....		
....		

3.3 VoiceManager初始化说明

```
VoiceManager.init(CarLife.receiver())
```

初始化录音相关，启动录音线程，参数为**CarLife.receiver()** 。车机MIC录音相关操作在SDK中处理，无需车厂管理。

3.4 显示分辨率的设置说明

```
CarLife.receiver().setDisplaySpec(displaySpec)
```

车机端发送给手机端支持的分辨率及帧率

参数说明

参数	含义
displaySpec	DisplaySpec：存储显示宽、高及分辨率的类

```
package com.baidu.carlife.sdk.sender.display

import android.content.Context
import com.baidu.carlife.sdk.util.annotations.DoNotStrip

@DoNotStrip
data class DisplaySpec (val context: Context, val width: Int, val height: Int, val
frameRate: Int) {
    val densityDpi: Int = context.resources.displayMetrics.densityDpi

    val ratio: Float = width.toFloat() / height
}
```

3.5 设置渲染Surface

```
CarLife.receiver().setSurface(mSurface)
```

参数说明

参数	含义
mSurface	Surface： 用于渲染投屏数据。使用sdk中提供的SurfaceView获得。

3.5.1 SurfaceView

SDK里面提供了两种SurfaceView：RemoteDisplayGLView 和 RemoteDisplayView，其中RemoteDisplayGLView可用与Android 5.0以上高版本，Android5.0以下低版本需使用 RemoteDisplayView。车厂接入时可选择其中一种使用。

```
class RemoteDisplayGLView @JvmOverloads constructor(context: Context, attrs:
AttributeSet? = null):GLSurfaceView(context, attrs)
class RemoteDisplayView @JvmOverloads constructor(context: Context, attrs:
AttributeSet? = null):SurfaceView(context, attrs)
```

示例

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/root_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".CarlifeActivity">

    <com.baidu.carlife.sdk.receiver.view.RemoteDisplayGLView
        android:id="@+id/video_surface_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/root_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".CarlifeActivity">

    <com.baidu.carlife.sdk.receiver.view.RemoteDisplayView
        android:id="@+id/video_surface_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

注：SurfaceView的宽高可以自定义，最后显示效果会根据手机端传输过来的分辨率进行计算显示在SurfaceView上

```
    override fun onVideoSizeChanged(videoWidth: Int, videoHeight: Int) {
        post {
            val ratio = videoWidth.toFloat() / videoHeight
            val (destWidth, destHeight) = ScaleUtils.inside(measuredWidth,
measuredHeight, ratio)
            layoutParams = layoutParams.apply {
                width = destWidth
                height = destHeight
            }
        }
    }
```

3.5.2 OnVideoSizeChangedListener

实现OnVideoSizeChangedListener接口并注册可以监听视频分辨率的更改。

```
package com.baidu.carlife.sdk.receiver;

import com.baidu.carlife.sdk.util.annotations.DoNotStrip;

@DoNotStrip
public interface OnVideoSizeChangedListener {
    void onVideoSizeChanged(int width, int height);
}
```

概要

Public methods

返回值	方法
void	onVideoSizeChanged(int width, int height) 手机端传输的视频分辨率发生改变时调用

注册监听

```
CarLife.receiver().addOnVideoSizeChangedListener(***)
```

3.6 读取本地配置

```
CarlifeConfUtil.getInstance().init()
```

这里是读取本地**/data/local/tmp/bdcf** 的配置文件，文件里面配置了每个车机端独有的配置，如车机渠道号等。在开始连接前需确保此配置文件读取成功完毕。

bdcf文件内容如下

```
#Channel Id: Baidu will provide the channel id to all OEM
20352101

# Audio Track Type 0:普通双音轨(路畅、华阳、飞哥) 1:单音轨(掌讯,仅用于测试) 2:定制双音轨-1(BYD、远峰) 3:定制双音轨-2（长安） 4：百度定制双音轨
AUDIO_TRACK_TYPE = 0

# 是否使用车机端MiC 0:使用车机端mic   1:使用手机端mic
VOICE_MIC = 0

# 是否支持唤醒
VOICE_WAKEUP = true

# 车机解码是否容易发生丢帧，打开后input2Decoder等待时间改为1000ms
NEED_MORE_DECODE_TIME = false

# 是否支持Carlife内置蓝牙电话
BLUETOOTH_INTERNAL_UI = false

# 是否支持蓝牙自动匹配
BLUETOOTH_AUTO_PAIR = false

# 是否使用透传方式
TRANSPARENT_SEND_TOUCH_EVENT = true

# 是否在每个触摸事件之前发送Action_Down
#SEND_ACTION_DOWN = false

# 是否使用车机端GPS
VEHICLE_GPS = false

# Focus Ui
FOCUS_UI = false

# 音频采样  0:跟随手机系统 1:48k
MEDIA_SAMPLE_RATE = 0

# Iphone手机的连接类 0:NCM 1:Wifi 2:NCM&Wifi(会显示切换的开关)
CONNECT_TYPE_IPHONE = 1

# iPhone手机通过USB的连接方  0：IPV6  1：IPV4  2：USB共享
IPHONE_USB_CONNECT_TYPE = 0

# iPhone手机通过USB—NCM连接映射在车机端网卡的名
```

```
IPHONE_NCM_ETHERNET_NAME = usb0

#Next song
#KEYCODE_SEEK_ADD = 23

#Previous song
#KEYCODE_SEEK_SUB = 22

#定制MD向HU传输音频流方 0：专用独立通道传输 1：蓝牙
AUDIO_TRANSMISSION_MODE=0

# 车机端支持的无线连接类型 0：不支持  1：热点  2：直连  3：都支持
CONFIG_WIRLESS_TYPE = 3

# 车机端支持的无线连接频率 0：2.4G  1：5G
CONFIG_WIRLESS_FREQUENCY = 1

# 是否支持蓝牙音频
CONFIG_BLUETOOTH_AUDIO = false
```

这里的Feature定制读取成功后，可以使用如下两个API接口设置给SDK：

```
CarLife.receiver().setFeatures(features: Map<String, Int>)
CarLife.receiver().setFeature(key: String, feature: Int)
```

注：设置的Feature如需同步给手机端，需在连接前完成。

3.7 Feature设置区分

3.7.1 setFeatures

```
    final override fun setFeatures(features: Map<String, Int>) {
        featureMap = HashMap(features)

        featureConfigChangeListeners.forEach { it.onFeatureConfigChanged() }
    }
```

setFeatures方法会重新创建Map，之前的设置将不复存在。

3.7.2 setFeature

```
    @Synchronized
    override fun setFeature(key: String, feature: Int) {
        featureMap[key] = feature

        featureConfigChangeListeners.forEach { it.onFeatureConfigChanged() }
    }
```

setFeature方法在原来的Map里面追加或覆盖相同的Key的value值。

四、连接

4.1 连接类型

SDK支持三种连接方式：USB、wifi热点、Wifi-Direct连接，车机端每一次连接只能选择其中一种连接方式。车机端初始化时可以默认设置其中一种，也可以中途更改。

4.1.1 连接选项

连接方式	要求	选项
USB连接	Android5.0以上支持AOA协议	必选
热点连接	Android5.0以上； wifi连接，工作在5GHz频段，支持802.11n\802.11ac	必选
无感连接	Wi-Fi模块：支持Wifi-Direct，并且工作在5GHz频段； 蓝牙：支持经典蓝牙，能和手机蓝牙进行配对连接.	可选

4.1.2 常量定义

位置：com.baidu.carlife.sdk.CarLifeContext

常量名称	常量值	备注
CONNECTION_TYPE_AOA	0x0002	USB
CONNECTION_TYPE_HOTSPOT	0x0005	WIFI热点
CONNECTION_TYPE_WIFIDIRECT	0x0009	Wifi-Direct

4.1.3 初始化配置

```
        val features = mapOf(
            ...
            FEATURE_CONFIG_CONNECT_TYPE to CarLifeContext.CONNECTION_TYPE_AOA
        )

        ....
        CarLife.init(
            this,
            "20029999",
            "12345678",
            features,
            CarlifeActivity::class.java,
            configs
        )
```

4.1.4 后续更改

```
      fun setConnectType(type:Int)
```

调用方式如下

```
CarLife.receiver().setConnectType(CarLifeContext.CONNECTION_TYPE_AOA)
```

注：车厂接入时必须支持USB和wifi热点两种，wifi p2p连接可选。

4.1.5 无感连接

无感连接即Wifi-Direct连接，若使用无感连接，需要注意下面几个配置：

配置	说明	Value
CONFIG_WIRLESS_TYPE	车机端支持的无线连接类型	2或3
CONFIG_WIRLESS_FREQUENCY	车机端支持的无线连接频率	1
CONFIG_WIFI_DIRECT_NAME	车机端WIFI直连的名称	自定义

这些配置可在Application初始化时配置在features中，也可以调用CarLife.receiver().setFeature()去配置。

注：无感连接时，车机需要把WIFI切换到5G频段，需要车机与手机先进行蓝牙绑定并连接上，车机端通过SDK封装的蓝牙socket通信与手机端建立连接，连接建立后，会把上面的配置信息发给手机端，手机端根据配置信息发起P2P直连逻辑。

在软件运行时，如果配置了上面得信息，如需查看配置是否正确，可以通过日志查看，日志如下：

```
CarLife_SDK: wifi_frequency: 1
CarLife_SDK: wirless_type: 3
CarLife_SDK: wifi_device_name: _ClfWfd_vehicle
```

其中，**_ClfWfd_vehicle**表示的是车机端Wifi直连的名字，可以先自行在手机端设置里面查看是否存在此Wifi直连的名字。如小米手机，可在**设置 -> WLAN -> 高级设置 -> WLAN直连** 里面查看车机端直连是否存在。**如果发现无感连接与手机端连接不成功，可先自行排查上面的信息是否有误，一定要确保日志里面传给手机端的WIFI直连名字跟手机系统设置里面传的名字一致。**

无感连接代码配置

部分车厂在接入SDK时反馈采用的Android系统非原生，会导致某些接口不可用。无感连接使用到了部分蓝牙接口与P2P相关接口，其中原生蓝牙接口需支持如下。

蓝牙方法：

方法名	类名	包名
isEnabled()	BluetoothAdapter	android.bluetooth
getBondedDevices()	BluetoothAdapter	android.bluetooth
createRfcommSocketToServiceRecord(UUID)	BluetoothDevice	android.bluetooth
isConnected()	BluetoothDevice	android.bluetooth
connect()	BluetoothSocket	android.bluetooth

蓝牙广播及状态：

广播名称	类名	包名
ACTION_STATE_CHANGED	BluetoothAdapter	android.bluetooth
ACTION_CONNECTION_STATE_CHANGED	BluetoothAdapter	android.bluetooth
EXTRA_STATE	BluetoothAdapter	android.bluetooth
STATE_OFF	BluetoothAdapter	android.bluetooth
EXTRA_CONNECTION_STATE	BluetoothAdapter	android.bluetooth
STATE_DISCONNECTED	BluetoothAdapter	android.bluetooth
STATE_CONNECTED	BluetoothAdapter	android.bluetooth
STATE_ON	BluetoothAdapter	android.bluetooth

4.2 获取本地文件配置

车机端开始连接前，可以读取一下本地的配置文件，把相应数据存储在缓存中（如：车机渠道号、车机wifi直连的名称、wifi的type类型、是否支持蓝牙音频等），读取完毕后，可以使用sdk上下文更改相应的设置，如下代码：

```
    fun init() {
        Logger.e(TAG, "++++++++++++++++++++Baidu Carlife Begin++++++++++++++++++++")

        // 根据配置文件设置相应的配置
        CarLife.receiver().setConfig(Configs.CONFIG_WIFI_DIRECT_NAME, "MI8") ---这里需要
配置车机端WIFI直连的名字
        CarLife.receiver().initStatisticsInfo(CommonParams.VEHICLE_CHANNEL, "12345678")
  ---这里重新设置车机渠道号，方便车厂在配置里面更改
        CarLife.receiver().setConfig(Configs.CONFIG_WIRLESS_TYPE,
CarlifeConfUtil.getInstance().getIntProperty(Configs.CONFIG_WIRLESS_TYPE))   ---无线连接
类型
        CarLife.receiver().setConfig(Configs.CONFIG_WIRLESS_FREQUENCY,
CarlifeConfUtil.getInstance().getIntProperty(Configs.CONFIG_WIRLESS_FREQUENCY)) ---无线
连接频率
        CarLife.receiver().setConfig(Configs.CONFIG_USE_BT_AUDIO,
CarlifeConfUtil.getInstance().getBooleanProperty(Configs.CONFIG_USE_BT_AUDIO))

    }
```

注：CarLife.receiver().initStatisticsInfo(CommonParams.VEHICLE_CHANNEL, "12345678") 此方法不建议使用，车机渠道号可在CarLife.init()中配置，这里的方法是方便车厂在配置文件配置渠道号，方便在不修改代码的情况下在配置文件中更改。

4.3 执行连接

配置文件设置完毕后，就可以执行连接过程了。

```
CarLife.receiver().connect();
```

注：连接过程中需要通过协议通道传递给手机端的配置都需要在连接前设置完成。

4.4 监听连接状态

4.4.1 ConnectionChangeListener

```
package com.baidu.carlife.sdk

import com.baidu.carlife.sdk.util.annotations.DoNotStrip

@DoNotStrip
interface ConnectionChangeListener {
    /**
     * 连接建立，分下面三种情况
     * 1 AOA连接建立
     * 2 ADB六条Socket建立成功
     * 3 Wifi六条Socket建立成功
```

```
     */
    @JvmDefault
    fun onConnectionAttached(context: CarLifeContext) {}

    /**
     * 在未detach的前提下，重新进行attach操作，会回调reattach
     */
    @JvmDefault
    fun onConnectionReattached(context: CarLifeContext) { }

    /**
     * 连接断开
     * 1 USB断开
     * 2 网络断开
     */
    @JvmDefault
    fun onConnectionDetached(context: CarLifeContext) {}

    /**
     * 协议校验成功
     * statistics info 校验成功时
     */
    @JvmDefault
    fun onConnectionEstablished(context: CarLifeContext) {}

    /**
     * 与设备协议版本不兼容
     * ProtocolVersionMatch 校验失败时
     */
    @JvmDefault
    fun onConnectionVersionNotSupprt(context: CarLifeContext) {}

    /**
     * 车机渠道号验证失败
     * CarlifeAuthenResult 校验失败时
     */
    @JvmDefault
    fun onConnectionAuthenFailed(context: CarLifeContext) {}
}
```

连接状态可以实现ConnectionChangeListener接口并注册来监听

概要

Public methods

返回值	方法
void	onConnectionAttached(context: CarLifeContext) 连接建立，分为三种情况：AOA连接建立、ADB六条Socket建立成功、wifi六条Socket建立成功
void	onConnectionReattached(context: CarLifeContext) 在未detach的前提下，重新进行attach操作，会回调reattach
void	onConnectionDetached(context: CarLifeContext) 连接断开：USB断开、网络断开
void	onConnectionEstablished(context: CarLifeContext) 协议校验成功时会回调此方法
void	onConnectionVersionNotSupprt(context: CarLifeContext) 与手机端CarLife 协议版本不兼容时会回调此方法
void	onConnectionAuthenFailed(context: CarLifeContext) 车机渠道号不合法时会回调此方法

注册监听

```
CarLife.receiver().registerConnectionChangeListener(this)
```

4.4.2 TransportListener

```
package com.baidu.carlife.sdk.internal.transport

import com.baidu.carlife.sdk.CarLifeContext
import com.baidu.carlife.sdk.ConnectionChangeListener
import com.baidu.carlife.sdk.internal.protocol.CarLifeMessage
import com.baidu.carlife.sdk.util.annotations.DoNotStrip

@DoNotStrip
interface TransportListener: ConnectionChangeListener {
    /**
     * called by transport when send a message
     * Avoid time-consuming operations
     * @return 是否继续停止分发
     */
    @JvmDefault
    fun onSendMessage(context: CarLifeContext, message: CarLifeMessage): Boolean {
        return false
```

```
    }

    /**
     * called by transport when receive a message
     * Avoid time-consuming operations
     * @return 是否继续停止分发
     */
    @JvmDefault
    fun onReceiveMessage(context: CarLifeContext, message: CarLifeMessage): Boolean {
        return false
    }
}
```

TransportListener继承了ConnectionChangeListener类，同时新增了收发协议的监听方法。如需要同时监听连接状态及收发协议的类可直接实现此接口并通过**CarLife.receiver()** 完成注册即可。

概要

Public methods

返回值	方法
Boolean	onSendMessage(context: CarLifeContext, message: CarLifeMessage) 车机端发送的每一条协议消息都可以在此监听
Boolean	onReceiveMessage(context: CarLifeContext, message: CarLifeMessage) 车机端收到的每一条协议消息都可以在此监听

注册监听

```
CarLife.receiver().registerTransportListener(this)
```

示例

在onReceiveMessage方法监听连接异常消息

```
    override fun onReceiveMessage(context: CarLifeContext, message: CarLifeMessage):
Boolean {
        when(message.serviceType) {
            ServiceTypes.MSG_CMD_CONNECT_EXCEPTION -> {
                val response= message.protoPayload as CarlifeConnectException
                handleConnectException(response)
            }
        }
        return false
    }
```

4.5 连接进度

连接进度可实现ConnectProgressListener接口并完成注册，可监听，进度值由SDK来控制。

4.5.1 ConnectProgressListener

```
package com.baidu.carlife.sdk.receiver

interface ConnectProgressListener {
    /**
     * 当前连接进度
     * @param progress 0 -> 100
     */
    fun onProgress(progress: Int)
}
```

注册监听

```
CarLife.receiver().addConnectProgressListener(this)
```

4.5.2 进度定义

进度值	含义
0	onConnectionAttached方法触发时 此时车机端开始发送MSG_CMD_HU_PROTOCOL_VERSION
30	收到手机端发送的MSG_CMD_PROTOCOL_VERSION_MATCH_STATUS消息，并且协议版本匹配正确 协议不匹配时触发onConnectionVersionNotSupprt方法，需提示升级手机端CarLife
70	onConnectionEstablished方法触发时 此处证明车机渠道号合法，当不合法时，触发onConnectionAuthenFailed方法，需提示该车机为非法车机
100	车机端发出MSG_CMD_VIDEO_ENCODER_START消息时，可认为连接成功 接下来只需要手机端发送投屏数据即可

4.6 连接异常处理

连接过程中，可能会出现以下几个问题：

- 手机端CarLife 版本未升级，导致与车机端协议版本不匹配，此时会回调onConnectionVersionNotSupprt方法，车机端可提示用户升级手机端CarLife后再进行连接操作；
- 车机端渠道号不合法，在手机端会验证车机端发送过来的渠道号，当不合法时，此时会回调onConnectionAuthenFailed方法，车机端可提示用户当前车机为非法车机；
- 当手机端为安装CarLife时，AOA连接时在手机会有系统框提示用户，但是车机端的现象时发送MSG_CMD_HU_PROTOCOL_VERSION协议一直没有响应，进度会一直在0%显示，此时这里车机端app可设置超时机制；
- 当CarLife手机端与车机端连接成功后，断开的情况下，此时会延时1秒后自动再次进行连接，由sdk自行处理。
- 其他。如有遇到，再补充。

4.7 断开连接

如果有的车厂在接入时需要有主动断开的逻辑，SDK里面也对外提供了断开接口。

```
fun disconnect()
```

调用很简单，如下代码：

```
CarLife.receiver().disconnect();
```

4.8 超时断开

CarLife SDK里面定义了超时断开，当车机端在8秒内没有收到来自手机端的消息，将会触发自动断开操作。断开时将有timeout的日志打印，如下所示：

```
ConnectionEstablishHandler heartbeat timeout
```

五、车机端功能定制

车机端的定制相关功能的支持情况是通过发送MSG_CMD_HU_FEATURE_CONFIG_RESPONSE消息并携带消息给到手机端的，这部分在SDK里面处理了，车机端app只需要在初始化CarLife前或者连接前通过设置相应的feature即可。

- CarLife的init方法执行前配置，然后传给init方法的features参数，如下：

```
        val features = mapOf(
            FEATURE_CONFIG_USB_MTU to 16 * 1024,
            FEATURE_CONFIG_FOCUS_UI to 1,
            FEATURE_CONFIG_I_FRAME_INTERVAL to 300
            FEATURE_CONFIG_VOICE_MIC to 1,
            FEATURE_CONFIG_VOICE_WAKEUP to 1
        )

        CarLife.init(
            ...,
            ...,
            ...,
            features,
            ...,
            ...)
```

- 开始连接前通过CarLife.receiver()设置feature参数，代码如下：

```
CarLife.receiver().setFeature(key: String, feature: Int)
CarLife.receiver().setFeatures(features: Map<String, Int>)
```

六、订阅消息

订阅相关消息涉及到手机端订阅车机端的消息、车机端订阅手机端的消息。车机端与手机端互为订阅者与发布者。

6.1 CarLifeSubscribable -- 发布者

手机端可以获取车载相关数据，包括车速、GPS定位信息、陀螺仪加速信息、油耗等。此时车机端可以称为发布者；当车机端向手机端订阅导航相关的辅助信息时，则此时手机端可以称为发布者。

SDK里面已定义好接口，提供信息的类只需实现该接口，并把实现了该发布者类通过**CarLife.receiver()** 添加即可。

```
package com.baidu.carlife.sdk
interface CarLifeSubscribable {
     val id: Int
     var supportFlag: Boolean

     fun subscribe()
     fun unsubscribe()
}
```

概要

Public methods

返回值	方法
void	subscribe() 在此开始发送数据
void	unsubscribe() 在此停止发送数据

Members

成员	含义
Id	Int: 信息模块ID，当前发布者提供的信息类型
supportFlag	Boolean: 是否支持订阅

6.1.1 车机端可以发布的信息

车身信息模块ID

```
enum ModuleID
{
    CAR_DATA_GPS=0;
    CAR_DATA_VELOCITY=1;
    CAR_DATA_GYROSCOPE=2;
    CAR_DATA_ACCELERATION=3;
    CAR_DATA_GEAR=4;
    CAR_DATA_OIL=5;
}
```

名称	含义
CAR_DATA_GPS	车身GPS模块
CAR_DATA_VELOCITY	车身速度模块
CAR_DATA_GYROSCOPE	车身陀螺仪模块
CAR_DATA_ACCELERATION	车身加速度模块
CAR_DATA_GEAR	车身档位模块
CAR_DATA_OIL	车身油量模块

注：见文档149页

6.1.2 手机端可以发布的消息

车机端需要使用CarLife里的导航模块提供的HUD信息或者电子眼信息时，可发起订阅。

手机端信息模块ID

```
enum ModuleID
{
    CARLIFE_DATA_TURNBYTURN=0;
    CARLFIE_DATA_ASSISTANTGUIDE=1;
}
```

名称	含义
CARLIFE_DATA_TURNBYTURN	CARLIFE_DATA_TURNBYTURN是简易诱导信息
CARLFIE_DATA_ASSISTANTGUIDE	CARLFIE_DATA_ASSISTANTGUIDE是辅助诱导信息

注：见152页

6.1.3 示例

实现车身GPS模块信息订阅

```
package com.baidu.carlifevehicle

import com.baidu.carlife.protobuf.CarlifeCarGpsProto
import com.baidu.carlife.protobuf.CarlifeNaviAssitantGuideInfoProto
import com.baidu.carlife.sdk.CarLifeContext
import com.baidu.carlife.sdk.CarLifeSubscribable
import com.baidu.carlife.sdk.Constants
import com.baidu.carlife.sdk.internal.protocol.CarLifeMessage
import com.baidu.carlife.sdk.internal.protocol.ServiceTypes
import com.baidu.carlife.sdk.sender.CarLife
import com.baidu.carlife.sdk.util.TimerUtils

class CarDataGPSSubscribable(private val context: CarLifeContext): CarLifeSubscribable
{
    override val id: Int = 0
    override var SupportFlag: Boolean = true

    override fun subscribe() {
        //开始给手机端发送GPS信息
        TimerUtils.schedule(subscribeGpsData, 1 * 1000, 2 * 1000)
    }

    override fun unsubscribe() {
        //停止给手机端发送GPS信息
        TimerUtils.stop(subscribeGpsData)
    }

    private val subscribeGpsData = Runnable {
        //开始给手机端发送当前汽车位置信息
        var message = CarLifeMessage.obtain(Constants.MSG_CHANNEL_CMD,
ServiceTypes.MSG_CMD_CAR_GPS)
        message.payload(
            CarlifeCarGpsProto.CarlifeCarGps.newBuilder()
                .setAntennaState(1)
                .setSignalQuality(2)
                .setLatitude(20)
                .setLongitude(10)
                .setHeight(1000)
                .setSpeed(80)
                .setHeading(90)
                .setYear(2021)
                .setMonth(7)
                .setDay(2)
                .setHrs(18)
                .setMin(30)
```

```
                .setSec(1)
                .setFix(1)
                .setHdop(2)
                .setPdop(1)
                .setVdop(3)
                .setSatsUsed(2)
                .setSatsVisible(2)
                .setHorPosError(4)
                .setVertPosError(5)
                .setNorthSpeed(6)
                .setEastSpeed(7)
                .setVertSpeed(8)
                .setTimeStamp(System.currentTimeMillis())
                .build())
        context.postMessage(message)
    }
}
```

注：各字段信息见文档144页

使用CarLife.receiver()添加

```
CarLife.receiver().addSubscribable(CarDataGPSSubscribable(CarLife.receiver()))
```

6.2 CarLifeSubscriber -- 订阅者

手机端向车机端订阅油耗等相关信息时，手机端为订阅者；车机端向手机端订阅导航相关简易诱导信息时，车机端为订阅者。

SDK里面已定义好接口，订阅者只需实现订阅接口接口，并把实现了该订阅者类通过**CarLife.receiver()** 添加即可。

```
package com.baidu.carlife.sdk

interface CarLifeSubscriber: TransportListener {
    val id: Int

    fun process(context: CarLifeContext, info: Any)
}
```

概要

Public methods

返回值	方法
void	process(context: CarLifeContext, info: Any) 处理订阅消息，如油耗信息

Members

成员	含义
id	Int: 信息模块ID 注：这里的信息模块ID同发布者是一样的

6.2.1 示例

手机端返回的导航简易诱导信息

```
package com.baidu.carlifevehicle

import android.os.Handler
import android.os.Looper
import com.baidu.carlife.protobuf.CarlifeNaviAssitantGuideInfoProto
import com.baidu.carlife.protobuf.CarlifeSubscribeMobileCarLifeInfoListProto
import com.baidu.carlife.protobuf.CarlifeSubscribeMobileCarLifeInfoProto
import com.baidu.carlife.sdk.CarLifeContext
import com.baidu.carlife.sdk.CarLifeSubscriber
import com.baidu.carlife.sdk.Constants
import com.baidu.carlife.sdk.internal.protocol.CarLifeMessage
import com.baidu.carlife.sdk.internal.protocol.ServiceTypes
import com.baidu.carlife.sdk.util.Logger

class AssistantGuideSubscriber(private val context: CarLifeContext) : CarLifeSubscriber
{
    private val TAG = "AssistantGuideSubscriber"

    override val id: Int = 1

    private var handler = Handler(Looper.getMainLooper())

    override fun process(context: CarLifeContext, info: Any) {
        var carLifeInfo = info as
CarlifeSubscribeMobileCarLifeInfoProto.CarlifeSubscribeMobileCarLifeInfo
```

```
        if (carLifeInfo.supportFlag) {
            snedCarlifeDataSubscribe(true)


            handler.postDelayed(Runnable {
                snedCarlifeDataSubscribe(false)
            }, 30 * 1000)
        }
    }

    override fun onReceiveMessage(context: CarLifeContext, message: CarLifeMessage):
Boolean {
        when (message.serviceType) {
            ServiceTypes.MSG_CMD_NAV_ASSISTANT_GUIDE_INFO ->
handleCarlifeDataSubscribeInfo(context, message)
        }

        return false
    }

    private fun snedCarlifeDataSubscribe(isStart: Boolean) {
        val infoListBuilder =

CarlifeSubscribeMobileCarLifeInfoListProto.CarlifeSubscribeMobileCarLifeInfoList.newBui
lder()
        val infoBuilder =
CarlifeSubscribeMobileCarLifeInfoProto.CarlifeSubscribeMobileCarLifeInfo.newBuilder()
        infoBuilder.moduleID = id
        infoBuilder.supportFlag = isStart
        infoListBuilder.addSubscribemobileCarLifeInfo(infoBuilder.build())

        infoListBuilder.cnt = infoListBuilder.subscribemobileCarLifeInfoCount

        val response = CarLifeMessage.obtain(
            Constants.MSG_CHANNEL_CMD,
            if (isStart) ServiceTypes.MSG_CMD_CARLIFE_DATA_SUBSCRIBE_START else
ServiceTypes.MSG_CMD_CARLIFE_DATA_SUBSCRIBE_STOP
        )
        response.payload(infoListBuilder.build())
        context.postMessage(response)
    }


    private fun handleCarlifeDataSubscribeInfo(context: CarLifeContext, message:
CarLifeMessage) {
        var response = message.protoPayload as?
CarlifeNaviAssitantGuideInfoProto.CarlifeNaviAssitantGuideInfo
        //开始处理移动设备Carlife发送过来的电子狗信息
        Logger.d(TAG, response?.toString())
```

```
    }
}
```

使用CarLife.receiver()添加

```
CarLife.receiver().addSubscriber(AssistantGuideSubscriber(CarLife.receiver()))
```

七、MIC录音规则

车机的MIC收音规则在Demo中有默认实现，如车厂需要使用默认，则需注意下面三个步骤：

- 初始化录音

```
VoiceManager.init(CarLife.receiver())
```

- 注册录音消息处理类

```
CarLife.receiver().registerTransportListener(VoiceMessageHandler())
```

- Activity生命周期传递

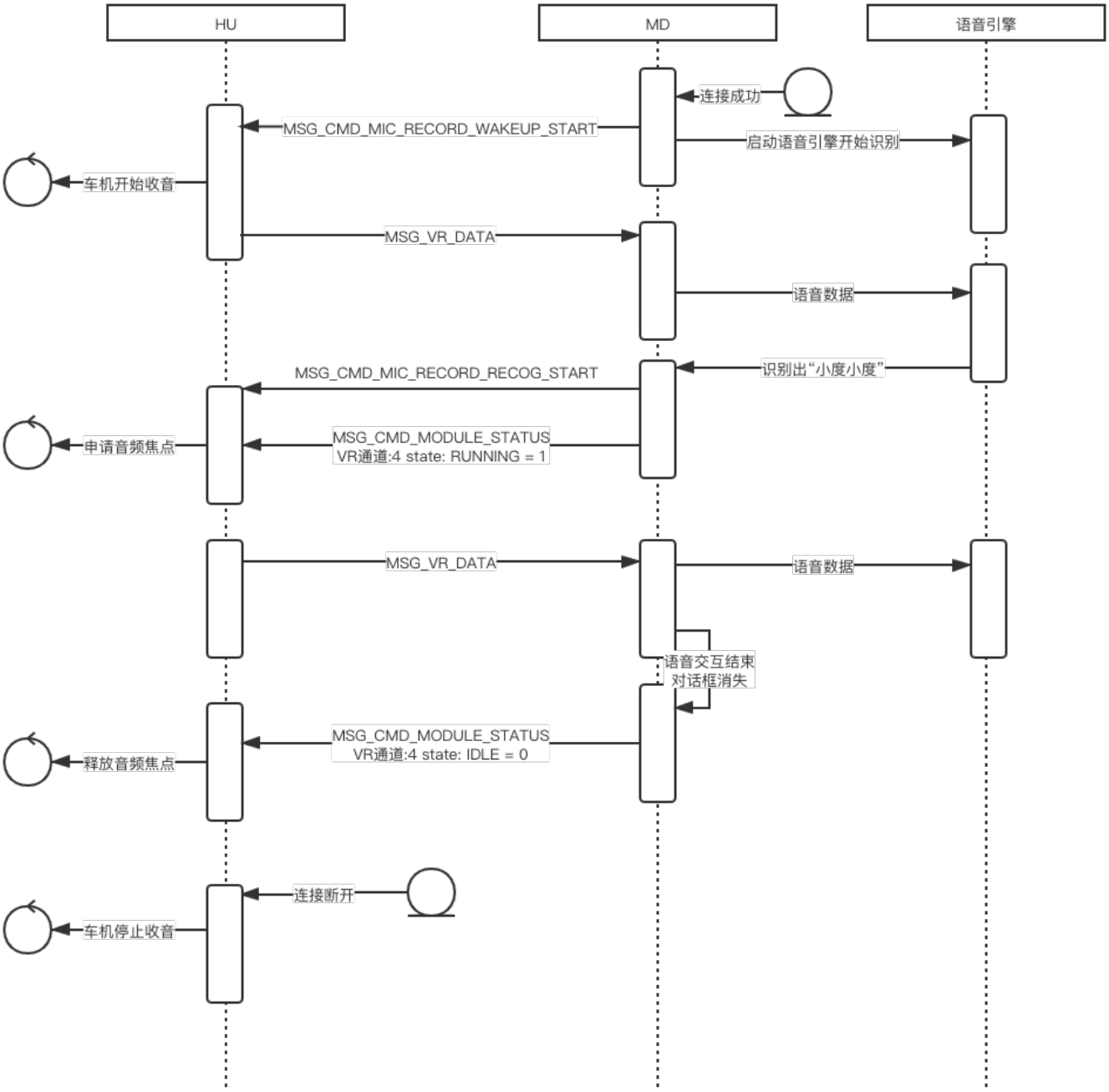
```
   override fun onStart() {
        super.onStart()
        VoiceManager.onActivityStart()
        ......
    }

    override fun onStop() {
        super.onStop()
        VoiceManager.onActivityStop()
        ......
    }
```

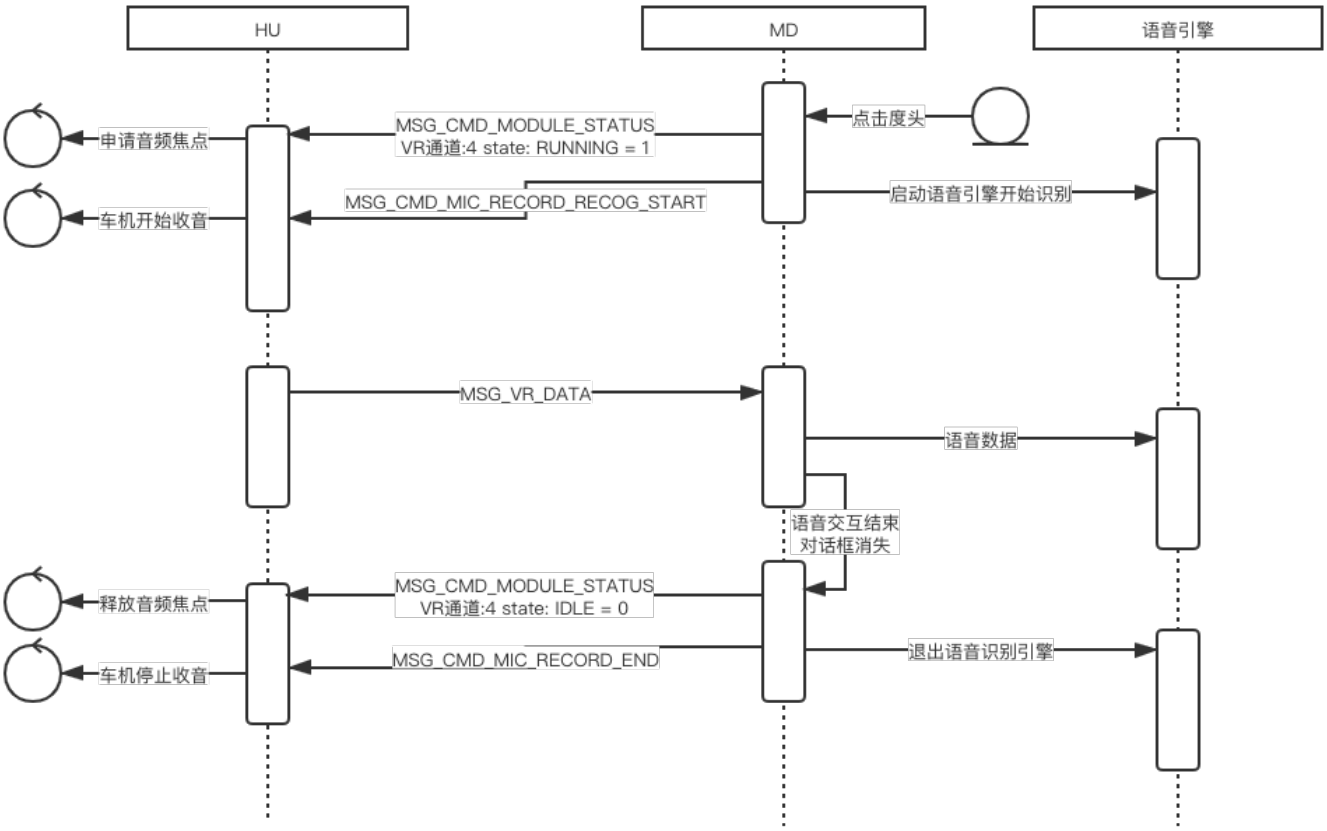
如果车厂需要自己实现一些定制化的MIC收音逻辑，可参考文档的规则。收音情况目前只支持以下四种规则，其中手机收音的无需车厂考虑。

录音	车机MIC	手机MIC
支持唤醒	7.1	
不支持唤醒	7.2	

7.1 支持唤醒的车机MIC规则



7.2 不支持唤醒的车机MIC规则



7.3 手机MIC录音

手机MIC的录音逻辑无需车机端处理收音相关消息。

八、模块状态

当移动设备端相关模块的状态改变时，就会发送给车机端，主要用于移动设备与车机互联时同步状态。涉及状态有：电话状态、导航状态、音乐状态、语音状态、连接状态、MIC状态、mediaPCM状态(仅用于IOS设备)、电子狗状态、cruise状态。

8.1 模块类型

模块名称	模块ID	枚举值	含义
电话	1	PHONE_STATUS_IDLE=0; PHONE_STATUS_INCOMING=1; PHONE_STATUS_OUTING=2;	
导航	2	NAVI_STATUS_IDLE=0; NAVI_STATUS_RUNNING =1;	
音乐	3	MUSIC_STATUS_IDLE=0; MUSIC_STATUS_RUNNING=1;	
语音	4	VR_STATUS_RECORD _IDLE =0; VR_STATUS_RECORD_RUNNING =1;	
连接	5	CONNECT_STATUS_ADB=1; CONNECT_STATUS_AOA=2; CONNECT_STATUS_NCM_ANDROID=3; CONNECT_STATUS_NCM_IOS =4; CONNECT_STATUS_WIFI=5;	
MIC	6	MIC_STATUS_USE_VEHICLE_MIC =0; MIC_STATUS_USE_MOBILE_MIC =1;	
MediaPCM	7	MEDIAPCM_STATUS_IDLE=0; MEDIAPCM_STATUS_RUNNING=1;	目前仅用于iOS设备。某些支持iPOD音乐播放的车机，当车机端CarLife切后台时，可以使用该命令暂停MD向HU 传输音乐PCM流，避免底层宽带不够用。
EDog	8	EDOG_STATUS_IDLE =0; EDOG_STATUS_RUNNING =1;	
Cruise	9	CRUISE_STATUS_IDLE =0; CRUISE_STATUS_RUNNING=1;	

8.2 状态处理

CarLifeSDK封装了一个抽象类，车机端APP如需监听某个模块的状态信息，直接继承该类即可，该抽象类为CarLifeModule.kt，如下：

```
package com.baidu.carlife.sdk

import com.baidu.carlife.sdk.internal.transport.TransportListener
import com.baidu.carlife.sdk.util.annotations.DoNotStrip

@DoNotStrip
abstract class CarLifeModule: TransportListener {
    protected var listener: ModuleStateChangeListener? = null

    abstract val id: Int

    open fun reader(): StreamReader {
```

```
        throw UnsupportedOperationException("module $id don't support read as stream")
    }

    var state: Int = 0
        set(value) {
            val oldState = field
            if (field != value) {
                field = value
                onModuleStateChanged(value, oldState)
            }
        }

    fun setStateChangeListener(listener: ModuleStateChangeListener?) {
        this.listener = listener
    }

    open fun onModuleStateChanged(newState: Int, oldState: Int) {
        listener?.onModuleStateChanged(this)
    }
}
```

实现该抽象类后，即使用**CarLife.receiver()** 注册即可，如下：

```
CarLife.receiver().addModule(****)
```

涉及模块状态的改变都会回调到onModuleStateChanged方法中。

8.2.1 CarLifeModule

CarLifeModule实现了TransportListener接口，可以监听连接状态的改变以及监听收发的协议消息。

概要

Public methods

返回值	方法
StreamReader	reader() VR数据相关操作
void	setStateChangeListener(listener: ModuleStateChangeListener?) 设置监听，如有需要的话，可以监听module状态的改变
void	onModuleStateChanged(newState: Int, oldState: Int) 状态改变时调用该方法

Members

成员	含义
listener	ModuleStateChangeListener：状态更改监听类
id	Int：模块ID，如音乐模块的值为3
state	Int：状态值ID

示例

监听电话状态。

```
package com.baidu.carlifevehicle.module

import com.baidu.carlife.sdk.CarLifeContext
import com.baidu.carlife.sdk.CarLifeModule
import com.baidu.carlife.sdk.Constants
import com.baidu.carlife.sdk.receiver.OnPhoneStateChangeListener
import com.baidu.carlife.sdk.util.Logger
import com.baidu.carlifevehicle.CarlifeActivity


import com.baidu.carlifevehicle.message.MsgHandlerCenter
import com.baidu.carlifevehicle.util.CommonParams

class PhoneModule(private val context: CarLifeContext,
                  private val activity: CarlifeActivity,
                  val callback: OnPhoneStateChangeListener)
    : CarLifeModule() {
    override val id: Int = Constants.PHONE_MODULE_ID

    override fun onModuleStateChanged(newState: Int, oldState: Int) {
        Logger.d(Constants.TAG, "newState: $newState, activity.mIsConnectException:
${activity.mIsConnectException}")
        // 电话状态车机端无需任何处理
        when(newState) {
            Constants.PHONE_STATUS_IDLE  -> {
                this@PhoneModule.callback.onStateChanged(false, false)
                if (activity.mIsConnectException) {

MsgHandlerCenter.dispatchMessage(CommonParams.MSG_MAIN_DISPLAY_MAIN_FRAGMENT)
                } else {CarlifeActivity
                    //通话结束后直接返回到投屏界面

MsgHandlerCenter.dispatchMessage(CommonParams.MSG_MAIN_DISPLAY_TOUCH_FRAGMENT)
                }
            }
```

```
            else -> {
                this@PhoneModule.callback.onStateChanged(true, newState ==
Constants.PHONE_STATUS_INCOMING)

MsgHandlerCenter.dispatchMessage(CommonParams.MSG_MAIN_DISPLAY_EXCEPTION_FRAGMENT)
            }
        }
    }
}
```

使用**CarLife.receiver()** 添加：

```
        mPhoneModule = PhoneModule(CarLife.receiver(), this, this)
        CarLife.receiver().addModule(mPhoneModule)
```

8.2.2 ModuleStateChangeListener

状态监听器

```
package com.baidu.carlife.sdk;

import com.baidu.carlife.sdk.util.annotations.DoNotStrip;

@DoNotStrip
public interface ModuleStateChangeListener {
    void onModuleStateChanged(CarLifeModule module);
}
```

实现该接口可以监听某个模块的状态变更，使用Module实例的setStateChangeListener方法设置即可。

概要

Public methods

返回值	方法
void	onModuleStateChanged(CarLifeModule module) 状态变更回调

8.3 模块控制

如车机端需要停止导航、停止音乐等操作，可以使用MSG_CMD_MODULE_CONTROL协议来操作。

8.3.1 暂停音乐

两种方式:

方式一、

```
            val message = CarLifeMessage.obtain(MSG_CHANNEL_CMD,
ServiceTypes.MSG_CMD_MODULE_CONTROL)
            message.payload(
                CarlifeModuleStatusProto.CarlifeModuleStatus
                    .newBuilder()
                    .setModuleID(Constants.MUSIC_MODULE_ID)
                    .setStatusID(Constants.MUSIC_STATUS_IDLE)
                    .build()
            )
            CarLife.receiver().postMessage(message)
```

方式二：即硬按键的处理方式：

```
            val message = CarLifeMessage.obtain(Constants.MSG_CHANNEL_TOUCH,
                     ServiceTypes.MSG_TOUCH_CAR_HARD_KEY_CODE)
            message.payload(
                        CarlifeCarHardKeyCodeProto.CarlifeCarHardKeyCode.newBuilder()
                                .setKeycode(KEYCODE_MEDIA_STOP)
                                .build());
            CarLife.receiver().postMessage(message);
```

8.3.2 停止导航

```
            val message = CarLifeMessage.obtain(MSG_CHANNEL_CMD,
ServiceTypes.MSG_CMD_MODULE_CONTROL)
            message.payload(
                CarlifeModuleStatusProto.CarlifeModuleStatus
                    .newBuilder()
                    .setModuleID(Constants.NAVI_MODULE_ID)
                    .setStatusID(Constants.NAVI_STATUS_IDLE)
                    .build()
            )
            CarLife.receiver().postMessage(message)
```

停止导航在硬按键的处理上只是调用返回健的处理方式：如下

```
                val message = CarLifeMessage.obtain(Constants.MSG_CHANNEL_TOUCH,
                        ServiceTypes.MSG_TOUCH_CAR_HARD_KEY_CODE)
                message.payload(
                       CarlifeCarHardKeyCodeProto.CarlifeCarHardKeyCode.newBuilder()
                               .setKeycode(KEYCODE_BACK)
                               .build());
               CarLife.receiver().postMessage(message);
```

九、CarLife上下文

SDK的所有接口调用都依赖CarLife上下文，所以这里列出其所有API接口，三方开发者可根据其实例进行调用。

9.1 CarLifeContext

CarLife上下文，通用方法封装，通过CarLifeContext可以发送消息、监听状态、加解密等等。

```
package com.baidu.carlife.sdk

import android.content.Context
import android.content.Intent
import android.content.SharedPreferences
import android.media.AudioManager
import android.os.Handler
import com.baidu.carlife.protobuf.*
import com.baidu.carlife.sdk.internal.protocol.CarLifeMessage
import com.baidu.carlife.sdk.internal.transport.TransportListener
import com.baidu.carlife.sdk.util.annotations.DoNotStrip
import java.io.File
import java.util.concurrent.ExecutorService

@DoNotStrip
interface CarLifeContext: TransportListener, ModuleStateChangeListener,
WirlessStatusListener {
    companion object {
        /**
         * 连接断开
         * 1 USB断开
         * 2 网络断开
         */
        const val CONNECTION_DETACHED    = 0
        /**
         * 连接建立，分下面三种情况
         * 1 AOA连接建立
         * 2 ADB六条Socket建立成功
         * 3 Wifi六条Socket建立成功
```

```
         */
        const val CONNECTION_ATTACHED    = 1

        /**
         * 重新attach，发生在
         * CONNECTION_ATTACHED -> CONNECTION_ATTACHED
         * CONNECTION_ESTABLISHED -> CONNECTION_ATTACHED
         */
        const val CONNECTION_REATTACH    = 2

        /**
         * 协议校验成功
         * statistics info 校验成功时
         */
        const val CONNECTION_ESTABLISHED = 3

        /**
         * 连接类型
         * aoa连接
         */
        const val CONNECTION_TYPE_AOA     = 0
        /**
         * 连接类型
         * wifi 热点连接
         */
        const val CONNECTION_TYPE_HOTSPOT = 1
        /**
         * 连接类型
         * wifi 直连
         */
        const val CONNECTION_TYPE_WIFIDIRECT = 2
    }

    val applicationContext: Context

    val osType: String

    val versionName: String

    val versionCode: Int

    var protocolVersion: Int

    var carlifeVersion: Int

    var isVersionMatch: Boolean

    var isVersionSupport: Boolean
```

```
// root dir for carlife stuff
val dataDir: File

// root dir for carlife caches
val cacheDir: File

// 存储设备相关信息，方便获取
var deviceInfo: CarlifeDeviceInfoProto.CarlifeDeviceInfo?

// 存储渠道信息，方便获取
var statisticsInfo: CarlifeStatisticsInfoProto.CarlifeStatisticsInfo?

// 获取连接状态
val connectionState: Int

// 获取连接类型
var connectionType: Int

var authResult: Boolean

val sharedPreferences: SharedPreferences

fun terminate()

// 设置支持的特性
fun setFeatures(features: Map<String, Int>)

// 设置支持的特性
fun setFeature(key: String, feature: Int)

// 获取支持的特性
fun getFeature(key: String, defaultConfig: Int): Int

// 枚举支持的特性
fun listFeatures(): List<CarlifeFeatureConfigProto.CarlifeFeatureConfig>

fun addFeatureConfigChangeListener(listener: FeatureConfigChangeListener)

fun removeFeatureConfigChangeListener(listener: FeatureConfigChangeListener)

// 异步发送消息
fun postMessage(message: CarLifeMessage)

// 同步发送消息
fun sendMessage(message: CarLifeMessage)

fun postMessage(channel: Int, serviceType: Int) {
    postMessage(CarLifeMessage.obtain(channel, serviceType))
}
```

```
fun sendMessage(channel: Int, serviceType: Int) {
    sendMessage(CarLifeMessage.obtain(channel, serviceType))
}

// 设置配置项
fun setConfig(key: String, config: Any)

// 删除配置项
fun removeConfig(key: String)

fun <T> getConfig(key: String, defaultConfig: T): T

fun <T> getConfig(key: String): T? {
    return getConfig(key, null)
}

// encryption
var isEncryptionEnabled: Boolean

// 对消息进行加密
fun encrypt(message: CarLifeMessage): CarLifeMessage

// 对消息进行解密
fun decrypt(message: CarLifeMessage): CarLifeMessage

// helpers
fun post(runnable: ()->Unit)

fun post(runnable: Runnable) {
    post { runnable.run() }
}

fun postDelayed(delayed: Long, runnable: ()->Unit)

fun postDelayed(runnable: Runnable, delayed: Long) {
    postDelayed(delayed) { runnable.run() }
}

// thread pools
fun main(): Handler

fun io(): ExecutorService

fun compute(): ExecutorService

// 请求音频焦点
fun requestAudioFocus(listener: AudioManager.OnAudioFocusChangeListener,
                      streamType: Int,
```

```
                        focusType: Int,
                        focusGrantDelayed: Boolean = false,
                        focusForceGrant: Boolean = false): Int

// 释放音频焦点
fun abandonAudioFocus(listener: AudioManager.OnAudioFocusChangeListener)

// Transport related
fun registerTransportListener(listener: TransportListener)

fun unregisterTransportListener(listener: TransportListener)

fun registerConnectionChangeListener(listener: ConnectionChangeListener)

fun unregisterConnectionChangeListener(listener: ConnectionChangeListener)

fun registerWirlessStatusListeners(listener: WirlessStatusListener)

fun unregisterWirlessStatusListeners(listener: WirlessStatusListener)

// configuration change related
fun notifyConfigurationChange(configuration: Int)

fun registerConfigurationListener(listener: ConfigurationChangeListener)

fun unregisterConfigurationListener(listener: ConfigurationChangeListener)

// 查找模块
fun findModule(id: Int): CarLifeModule?

// 枚举模块
fun listModule(): List<CarLifeModule>

// 添加模块
fun addModule(module: CarLifeModule)

// 删除模块
fun removeModule(module: CarLifeModule)

// 查找订阅者
fun findSubscriber(id: Int): CarLifeSubscriber?

// 添加订阅者
fun addSubscriber(subscriber: CarLifeSubscriber)

// 删除订阅者
fun removeSubscriber(subscriber: CarLifeSubscriber)
```

```
    fun listSubscriber():
List<CarlifeSubscribeMobileCarLifeInfoProto.CarlifeSubscribeMobileCarLifeInfo>

    // 查找被订阅者
    fun findSubscribable(id: Int): CarLifeSubscribable?

    // 添加被订阅者
    fun addSubscribable(subscribable: CarLifeSubscribable)

    // 删除被订阅者
    fun removeSubscribable(subscribable: CarLifeSubscribable)


    fun listSubscribable(): List<CarlifeVehicleInfoProto.CarlifeVehicleInfo>

    fun onNewIntent(intent: Intent)

    // 判断当前连接是否成功建立，对应Established
    fun isConnected(): Boolean

    fun isAttached(): Boolean

    fun getConnectType(): Int

    /**
     * 与设备建立连接
     * @CallSuper
     */
    @JvmDefault
    fun onConnectionAttached() {
        onConnectionAttached(this)
    }

    /**
     *  与设备断开连接
     *  @CallSuper
     */
    @JvmDefault
    fun onConnectionDetached() {
        onConnectionDetached(this)
    }

    /**
     * 与设备经过握手
     * @CallSuper
     */
    @JvmDefault
    fun onConnectionEstablished() {
        onConnectionEstablished(this)
```

```
    }

    /**
     *  与设备协议版本不兼容
     *  @CallSuper
     */
    @JvmDefault
    fun onConnectionVersionNotSupprt() {
        onConnectionVersionNotSupprt(this)
    }

    /**
     *  车机渠道号验证失败
     *  @CallSuper
     */
    @JvmDefault
    fun onConnectionAuthenFailed() {
        onConnectionAuthenFailed(this)
    }
}
```

9.2 CarLifeReceiver

CarLifeReceiver车机端抽象接口，具备车机端的特有能力，反控事件发送、语音收音、音频播放与焦点管理等。

```
package com.baidu.carlife.sdk.receiver

import android.app.Activity
import android.view.MotionEvent
import android.view.Surface
import com.baidu.carlife.sdk.CarLifeContext
import com.baidu.carlife.sdk.sender.display.DisplaySpec
import com.baidu.carlife.sdk.util.annotations.DoNotStrip
import java.io.File

@DoNotStrip
interface CarLifeReceiver: CarLifeContext {

    // 连接成功时，需要拉起的Activity
    val activityClass: Class<out Activity>
    /**
     * 添加连接进度监听器
     */
    fun addConnectProgressListener(listener: ConnectProgressListener)

    /**
```

```
     * 删除连接进度监听器
     */
    fun removeConnectProgressListener(listener: ConnectProgressListener)

    // 设置文件接收监听器
    fun setFileTransferListener(listener: FileTransferListener?)

    // 设置视频帧的渲染参数(屏幕宽高、帧率)
    fun setDisplaySpec(displaySpec: DisplaySpec)

    // 创建好SurfaceView之后，调用此方法传递Surface，用于构建MediaCodec，解码渲染
    // 传递空的话，停止渲染，非空开始渲染，主线程调用
    fun setSurface(surface: Surface?)

    fun setSurfaceRequestCallback(callback: SurfaceRequestCallback?)

    fun onSurfaceSizeChanged(width: Int, height: Int)

    fun addOnVideoSizeChangedListener(listener: OnVideoSizeChangedListener)

    fun removeOnVideoSizeChangedListener(listener: OnVideoSizeChangedListener)

    // Activity 生命周期回调
    fun onActivityStarted()

    fun onActivityStopped()

    // 传递反控事件
    fun onTouchEvent(event: MotionEvent)

    // 传递硬按键消息
    fun onKeyEvent(keyCode: Int)

    fun initStatisticsInfo(channel: String, cuid: String)

    fun connect()
}
```

十、蓝牙音频

车机端2.0版本规定车厂接入后将使用蓝牙音频，使用蓝牙音频后，手机端将不会把音频数据传输到车机端，直接走蓝牙通道播放声音，手机和车机可以共同控制音频的音量大小。

10.1 蓝牙音频优点

1、车厂无需处理pcm音频流数据，不用关心CarLife相关的音频焦点，减少研发工作；

2、手机上的CarLife音频统一走蓝牙通道到车机端，车机手机可以共同控制音频音量大小；

3、手机上三方应用的音频播放可以从车机蓝牙播放出来；

10.2 注意事项

1、手机车机蓝牙未连接时，音频将从手机播放；

10.3 开启蓝牙音频

蓝牙音频配置开关在SDK里面有定义如下：

```
public static final String FEATURE_CONFIG_AUDIO_TRANSMISSION_MODE =
"AUDIO_TRANSMISSION_MODE";
```

将其Key的值设置为1时则会开启蓝牙音频，设置方式可以在初始化时设置，如下：

```
private fun initReceiver() {
        ......

        val features = mapOf(
            ......
            FEATURE_CONFIG_AUDIO_TRANSMISSION_MODE to 1
        )

        val configs = mapOf(
            ......
        )

        CarLife.init(
            this,
            "",
            "",
            features,
            CarlifeActivity::class.java,
            configs)
        VoiceManager.init(CarLife.receiver())
    }
```

也可以通过setFeature方法设置，如下：

```
CarLife.receiver().setFeature(FEATURE_CONFIG_AUDIO_TRANSMISSION_MODE, 1)
```

十一、aac编码压缩

车机端2.0优化了音频带宽占用问题，支持aac编码压缩，可以减少80%音频数据量。车机端配置支持aac编码压缩后手机端carlife会将音频压缩后传递至车机端解码播放。

11.1 aac编码压缩优点

1、减少网络带宽的占用，提高carlife连接稳定性。

11.2 注意事项

1、aac时通过硬编硬解，需要车机端解码器支持，如遇硬件不兼容问题，需要关闭aac编码开关；

11.3 开启aac编码

aac编码开关在SDK里面的定义如下：

```
public static final String FEATURE_CONFIG_AAC_SUPPORT = "AAC_SUPPORT";
```

将其Key的值设置为1时则表示车机端支持aac解码，设置方式可以在初始化时设置，如下：

```
private fun initReceiver() {
        ......

        val features = mapOf(
            ......
            FEATURE_CONFIG_AAC_SUPPORT to 1
        )

        val configs = mapOf(
            ......
        )

        CarLife.init(
            this,
            "",
            "",
            features,
            CarlifeActivity::class.java,
            configs)
        VoiceManager.init(CarLife.receiver())
    }
```

同蓝牙音频一样，还可使用setFeature设置。

其他

SDK日志路径

sdcard/Android/data/应用包名/files/log/*，SDK日志关键字"CarLife_SDK"

ServiceTypes 常量定义

```
package com.baidu.carlife.sdk.internal.protocol

import com.baidu.carlife.sdk.util.annotations.DoNotStrip

@DoNotStrip
object ServiceTypes {
    // 初始化时交互的信息
    const val MSG_CMD_HU_PROTOCOL_VERSION = 0x00018001
    const val MSG_CMD_PROTOCOL_VERSION_MATCH_STATUS = 0x00010002
    const val MSG_CMD_HU_INFO = 0x00018003
    const val MSG_CMD_MD_INFO = 0x00010004
    const val MSG_CMD_HU_BT_OOB_INFO = 0x00018005
    const val MSG_CMD_MD_BT_OOB_INFO = 0x00010006

    // 视频相关消息
    const val MSG_CMD_VIDEO_ENCODER_INIT = 0x00018007
    const val MSG_CMD_VIDEO_ENCODER_INIT_DONE = 0x00010008
    const val MSG_CMD_VIDEO_ENCODER_START = 0x00018009
    const val MSG_CMD_VIDEO_ENCODER_PAUSE = 0x0001800A
    const val MSG_CMD_VIDEO_ENCODER_RESET = 0x0001800B
    const val MSG_CMD_VIDEO_ENCODER_FRAME_RATE_CHANGE = 0x0001800C
    const val MSG_CMD_VIDEO_ENCODER_FRAME_RATE_CHANGE_DONE = 0x0001000D
    const val MSG_CMD_MD_VIDEO_ENCODER_REQ = 0x0001000F

    // 音频相关消息
    const val MSG_CMD_PAUSE_MEDIA = 0x0001800E

    // 车载数据消息订阅机制
    const val MSG_CMD_CAR_DATA_SUBSCRIBE_REQ = 0x00010031
    const val MSG_CMD_CAR_DATA_SUBSCRIBE_RSP = 0x00018032
    const val MSG_CMD_CAR_DATA_START_REQ = 0x00010033
    const val MSG_CMD_CAR_DATA_STOP_REQ = 0x00010034

    // 车载信息
    const val MSG_CMD_CAR_VELOCITY = 0x0001800F
    const val MSG_CMD_CAR_GPS = 0x00018010
    const val MSG_CMD_CAR_GYROSCOPE = 0x00018011
```

```
const val MSG_CMD_CAR_ACCELERATION = 0x00018012
const val MSG_CMD_CAR_OIL = 0x00018013
// 档位状态
const val MSG_CMD_CAR_GEAR = 0x00018029
// 空挡
const val GEAR_NEUTRAL = 1
// 驻车档
const val GEAR_PARK = 2
// 动力档
const val GEAR_DRIVE = 3
// 低速挡
const val GEAR_LOW = 4
// 倒车档
const val GEAR_REVERSE = 5

// 电话状态
const val MSG_CMD_TEL_STATE_CHANGE_INCOMING = 0x00010014
const val MSG_CMD_TEL_STATE_CHANGE_OUTGOING = 0x00010015
const val MSG_CMD_TEL_STATE_CHANGE_IDLE = 0x00010016
const val MSG_CMD_TEL_STATE_CHANGE_INCALLING = 0x00010017

// 移动设备端状态
const val MSG_CMD_SCREEN_ON = 0x00010018
const val MSG_CMD_SCREEN_OFF = 0x00010019
const val MSG_CMD_SCREEN_USER_PRESENT = 0x0001001A
const val MSG_CMD_FOREGROUND = 0x0001001B
const val MSG_CMD_BACKGROUND = 0x0001001C

// 启动模式
const val MSG_CMD_LAUNCH_MODE_NORMAL = 0x0001801D
const val MSG_CMD_LAUNCH_MODE_PHONE = 0x0001801E
const val MSG_CMD_LAUNCH_MODE_MAP = 0x0001801F
const val MSG_CMD_LAUNCH_MODE_MUSIC = 0x00018020
const val MSG_CMD_GO_TO_DESKTOP = 0x00010021

// 语音相关
const val MSG_CMD_MIC_RECORD_WAKEUP_START = 0x00010022
const val MSG_CMD_MIC_RECORD_WAKEUP_DONE = 0x00018022
const val MSG_CMD_MIC_RECORD_END = 0x00010023
const val MSG_CMD_MIC_RECORD_RECOG_START = 0x00010024
const val MSG_CMD_MIC_RECORD_RECOG_DONE = 0x00018024

const val MSG_CMD_GO_TO_FOREGROUND = 0x00018025
const val MSG_CMD_MODULE_STATUS = 0x00010026
const val MSG_CMD_STATISTIC_INFO = 0x00018027
const val MSG_CMD_MODULE_CONTROL = 0x00018028

// 音乐相关
const val MSG_CMD_MEDIA_INFO = 0x00010035
```

```
const val MSG_CMD_MEDIA_PROGRESS_BAR = 0x00010036

const val MSG_CMD_CONNECT_EXCEPTION = 0x00010037
const val MSG_CMD_REQUEST_GO_TO_FOREGROUND = 0x00010038

// 点击反馈发消息到车机
const val MSG_CMD_UI_ACTION_SOUND = 0x00010039

// 蓝牙电话相关的命令消息
const val MSG_CMD_BT_HFP_REQUEST = 0x00010040
const val MSG_CMD_BT_HFP_INDICATION = 0x00018041
const val MSG_CMD_BT_HFP_CONNECTION = 0x00018042

// 车机端订阅手机端信息的消息
const val MSG_CMD_MD_CARLIFE_DATA_REQ = 0x00010043
const val MSG_CMD_CARLIFE_DATA_SUBSCRIBE = 0x00018043
const val MSG_CMD_CARLIFE_DATA_SUBSCRIBE_DONE = 0x00010044
const val MSG_CMD_HU_CARLIFE_DATA_SUBSCRIBE_DONE_RESPONSE = 0x00018044
const val MSG_CMD_CARLIFE_DATA_SUBSCRIBE_START = 0x00018045
const val MSG_CMD_CARLIFE_DATA_SUBSCRIBE_STOP = 0x00018046
const val MSG_CMD_NAV_ASSISTANT_GUIDE_INFO = 0x00018047
const val MSG_CMD_NAV_NEXT_TURN_INFO = 0x00010030


// 车机对手机进行安全认证
const val MSG_CMD_HU_AUTH_REQUEST = 0x00018048
const val MSG_CMD_MD_AUTH_RESPONSE = 0x00010049
const val MSG_CMD_HU_AUTH_RESULT = 0x0001804A
const val MSG_CMD_MD_AUTH_RESULT = 0x0001004B
const val MSG_CMD_MD_AUTH_RESULT_RESPONSE = 0x0001804C

const val MSG_CMD_START_BT_AUTO_PAIR_REQUEST = 0x0001004D
const val MSG_CMD_BT_HFP_RESPONSE = 0x0001804E
const val MSG_CMD_BT_HFP_STATUS_REQUEST = 0x0001004F
const val MSG_CMD_BT_HFP_STATUS_RESPONSE = 0x00018050

// 功能定制相关的命令消息
const val MSG_CMD_MD_FEATURE_CONFIG_REQUEST = 0x00010051
const val MSG_CMD_HU_FEATURE_CONFIG_RESPONSE = 0x00018052

const val MSG_CMD_BT_START_IDENTIFY_REQ = 0x00018053
const val MSG_CMD_BT_IDENTIFY_RESULT_IND = 0x00010054

// 错误码统计信息
const val MSG_CMD_ERROR_CODE = 0x00018055

const val MSG_CMD_VIDEO_ENCODER_JPEG = 0x00018056
const val MSG_CMD_VIDEO_ENCODER_JPEG_ACK = 0x00010057
```

```
const val MSG_CMD_BT_HFP_CALL_STATUS_COVER = 0x00010058

const val MSG_CMD_MD_EXIT = 0x00010059

// 车控相关
const val MSG_CMD_VEHICLE_CONTROL_INFO = 0x00018061
const val MSG_CMD_VEHICLE_CONTROL = 0x0001006F
const val MSG_VEHICLE_CONTROL_TYPE_GET = 0x0001
const val MSG_VEHICLE_CONTROL_TYPE_POST = 0x0002
const val MSG_VEHICLE_CONTROL_VALUE_TYPE_STRING = 0x00100000
const val MSG_VEHICLE_CONTROL_VALUE_TYPE_BOOLEAN = 0x00200000
const val MSG_VEHICLE_CONTROL_VALUE_TYPE_INT32 = 0x00400000
const val MSG_VEHICLE_CONTROL_VALUE_TYPE_INT64 = 0x00500000
const val MSG_VEHICLE_CONTROL_VALUE_TYPE_FLOAT = 0x00600000
const val MSG_VEHICLE_CONTROL_VALUE_TYPE_BYTES = 0x00700000
const val MSG_VEHICLE_CONTROL_ID_GET_ALL_STATUS = 0x0001
const val MSG_VEHICLE_CONTROL_ID_GET_BASE_STATUS = 0x0002
const val MSG_VEHICLE_CONTROL_ID_GET_HVAC_STATUS = 0x0003
const val MSG_VEHICLE_CONTROL_ID_GET_MIRROR_STATUS = 0x0004
const val MSG_VEHICLE_CONTROL_ID_GET_SEAT_STATUS = 0x0005
const val MSG_VEHICLE_CONTROL_ID_GET_WINDOW_STATUS = 0x0006

const val MSG_VEHICLE_CONTROL_GET_ALL_STATUS = 0x0001
const val MSG_VEHICLE_CONTROL_GET_BASE_STATUS = 0x0002
const val MSG_VEHICLE_CONTROL_GET_HVAC_STATUS = 0x0003
const val MSG_VEHICLE_CONTROL_GET_MIRROR_STATUS = 0x0004
const val MSG_VEHICLE_CONTROL_GET_SEAT_STATUS = 0x0005
const val MSG_VEHICLE_CONTROL_GET_WINDOW_STATUS = 0x0006

const val MSG_VEHICLE_CONTROL_ID_WINDOW_MOVE = 0x0BC1
const val MSG_VEHICLE_CONTROL_ID_WINDOW_POS = 0x0BC0
const val MSG_VEHICLE_CONTROL_ID_HVAC_POWER_ON = 0x0510
const val MSG_VEHICLE_CONTROL_ID_HVAC_TEMPERATURE_SET = 0x0503
const val MSG_VEHICLE_CONTROL_ID_HVAC_WIND_SET = 0x0504
const val MSG_VEHICLE_CONTROL_ID_HVAC_WIND_TYPE_SET = 0x0505
const val MSG_VEHICLE_CONTROL_ID_SEAT_FORE_AFT_MOVE = 0x0B86
const val MSG_VEHICLE_CONTROL_ID_SEAT_BACKREST_ANGLE_MOVE = 0x0B88
const val MSG_VEHICLE_CONTROL_ID_MIRROR_ON = 0x0B46
const val MSG_VEHICLE_CONTROL_ID_MIRROR_Y_POS = 0x0B42
const val MSG_VEHICLE_CONTROL_ID_MIRROR_Y_MOVE = 0x0B43
const val MSG_VEHICLE_CONTROL_ID_MIRROR_Z_POS = 0x0B40
const val MSG_VEHICLE_CONTROL_ID_MIRROR_Z_MOVE = 0x0B41

const val MSG_VEHICLE_CONTROL_AREA_ID_GLOBAL = 0x01000000
const val MSG_VEHICLE_CONTROL_AREA_ID_ZONE = 0x02000000
const val MSG_VEHICLE_CONTROL_AREA_ID_WINDOW = 0x03000000
const val MSG_VEHICLE_CONTROL_AREA_ID_MIRROR = 0x04000000
const val MSG_VEHICLE_CONTROL_AREA_ID_SEAT = 0x05000000
const val MSG_VEHICLE_CONTROL_AREA_ID_DOOR = 0x06000000
```

```
const val MSG_VEHICLE_CONTROL_AREA_WINDOW_FRONT_WINDSHIELD = 0x0001
const val MSG_VEHICLE_CONTROL_AREA_WINDOW_REAR_WINDSHIELD = 0x0002
const val MSG_VEHICLE_CONTROL_AREA_WINDOW_ROOF_TOP = 0x0004
const val MSG_VEHICLE_CONTROL_AREA_WINDOW_ROW_1_LEFT = 0x0010
const val MSG_VEHICLE_CONTROL_AREA_WINDOW_ROW_1_RIGHT = 0x0020
const val MSG_VEHICLE_CONTROL_AREA_WINDOW_ROW_2_LEFT = 0x0100
const val MSG_VEHICLE_CONTROL_AREA_WINDOW_ROW_2_RIGHT = 0x0200
const val MSG_VEHICLE_CONTROL_AREA_WINDOW_ROW_3_LEFT = 0x1000
const val MSG_VEHICLE_CONTROL_AREA_WINDOW_ROW_3_RIGHT = 0x2000
const val MSG_VEHICLE_CONTROL_AREA_WINDOW_ALL = 0x3000

const val MSG_VEHICLE_CONTROL_AREA_SEAT_ROW_1_LEFT = 0x0001
const val MSG_VEHICLE_CONTROL_AREA_SEAT_ROW_1_CENTER = 0x0002
const val MSG_VEHICLE_CONTROL_AREA_SEAT_ROW_1_RIGHT = 0x0004
const val MSG_VEHICLE_CONTROL_AREA_SEAT_ROW_2_LEFT = 0x0010
const val MSG_VEHICLE_CONTROL_AREA_SEAT_ROW_2_CENTER = 0x0020
const val MSG_VEHICLE_CONTROL_AREA_SEAT_ROW_2_RIGHT = 0x0040
const val MSG_VEHICLE_CONTROL_AREA_SEAT_ROW_3_LEFT = 0x0100
const val MSG_VEHICLE_CONTROL_AREA_SEAT_ROW_3_CENTER = 0x0200
const val MSG_VEHICLE_CONTROL_AREA_SEAT_ROW_3_RIGHT = 0x0400

const val MSG_VEHICLE_CONTROL_AREA_MIRROR_DRIVER_LEFT = 0x00000001
const val MSG_VEHICLE_CONTROL_AREA_MIRROR_DRIVER_RIGHT = 0x00000002
const val MSG_VEHICLE_CONTROL_AREA_MIRROR_DRIVER_CENTER = 0x00000004


const val MSG_VEHICLE_CONTROL_INFO_SUPPORT_ENABLE = 0x0001
const val MSG_VEHICLE_CONTROL_INFO_SUPPORT_DISABLE = 0x0002

const val MSG_VEHICLE_CONTROL_INFO_BASE_POWER_STATUS = 0x0001
const val MSG_VEHICLE_CONTROL_INFO_BASE_DRIVE_STATUS = 0x0002
const val MSG_VEHICLE_CONTROL_INFO_BASE_SPEED_TYPE = 0x0003
const val MSG_VEHICLE_CONTROL_INFO_BASE_SPEED_VALUE = 0x0004

const val MSG_VEHICLE_CONTROL_INFO_HVAC_POWER_STATUS = 0x0005
const val MSG_VEHICLE_CONTROL_INFO_HVAC_TEMPERATURE_TYPE = 0x0006
const val MSG_VEHICLE_CONTROL_INFO_HVAC_TEMPERATURE_VALUE = 0x0007
const val MSG_VEHICLE_CONTROL_INFO_HVAC_WIND_TYPE = 0x0008
const val MSG_VEHICLE_CONTROL_INFO_HVAC_WIND_MAX_LEVEL = 0x0009
const val MSG_VEHICLE_CONTROL_INFO_HVAC_WIND_SPEED = 0x000A

const val MSG_VEHICLE_CONTROL_INFO_MIRROR_STATUS = 0x000B

const val MSG_VEHICLE_CONTROL_INFO_VALUE_TYPE_STRING = 0x00100000
const val MSG_VEHICLE_CONTROL_INFO_VALUE_TYPE_INT32 = 0x00200000
const val MSG_VEHICLE_CONTROL_INFO_VALUE_TYPE_INT64 = 0x00300000
const val MSG_VEHICLE_CONTROL_INFO_VALUE_TYPE_FLOAT = 0x00400000
```

```
const val MSG_VEHICLE_CONTROL_INFO_VALUE_ON = 0x0001
const val MSG_VEHICLE_CONTROL_INFO_VALUE_OFF = 0x0002

const val MSG_VEHICLE_CONTROL_INFO_VALUE_KMH = 0x0001
const val MSG_VEHICLE_CONTROL_INFO_VALUE_MPH = 0x0002

const val MSG_VEHICLE_CONTROL_INFO_VALUE_CENTIGRADE = 0x0001
const val MSG_VEHICLE_CONTROL_INFO_VALUE_FAHRENHEIT = 0x0002

const val MSG_VEHICLE_CONTROL_INFO_VALUE_AUTO = 0x0001
const val MSG_VEHICLE_CONTROL_INFO_VALUE_MANUAL = 0x0002

// Video通道相关消息
const val MSG_VIDEO_DATA = 0x00020001

const val MSG_VIDEO_HEARTBEAT = 0x00020002


// Media通道相关消息
const val MSG_MEDIA_INIT = 0x00030001
const val MSG_MEDIA_STOP = 0x00030002
const val MSG_MEDIA_PAUSE = 0x00030003
const val MSG_MEDIA_RESUME_PLAY = 0x00030004
const val MSG_MEDIA_SEEK_TO = 0x00030005
const val MSG_MEDIA_DATA = 0x00030006

// TTS通道相关消息
const val MSG_NAV_TTS_INIT = 0x00040001
const val MSG_NAV_TTS_END = 0x00040002
const val MSG_NAV_TTS_DATA = 0x00040003

// VR通道相关消息
const val MSG_VR_DATA = 0x00058001
const val MSG_VR_AUDIO_INIT = 0x00050002
const val MSG_VR_AUDIO_DATA = 0x00050003
const val MSG_VR_AUDIO_STOP = 0x00050004
const val MSG_VR_MODULE_STATUS = 0x00050005
const val MSG_VR_AUDIO_INTERRUPT = 0x00050006

// Touch通道相关消息
const val MSG_TOUCH_ACTION = 0x00068001
const val MSG_TOUCH_ACTION_DOWN = 0x00068002
const val MSG_TOUCH_ACTION_UP = 0x00068003
const val MSG_TOUCH_ACTION_MOVE = 0x00068004
const val MSG_TOUCH_SINGLE_CLICK = 0x00068005
const val MSG_TOUCH_DOUBLE_CLICK = 0x00068006
const val MSG_TOUCH_LONG_PRESS = 0x00068007
const val MSG_TOUCH_UI_ACTION_SOUND = 0x00060009
const val MSG_TOUCH_UI_ACTION_BEGIN = 0x0006800A
```

```
    const val MSG_TOUCH_ACTION_POINTER_DOWN = 0x0006800B
    const val MSG_TOUCH_ACTION_POINTER_UP = 0x0006800C
    const val MSG_TOUCH_ACTION_OTHER_POINTER_UP = 0x0006800D

    const val MSG_TOUCH_CAR_HARD_KEY_CODE = 0x00068008
    // 硬按键消息
    const val KEYCODE_HOME = 0x00000001
    const val KEYCODE_PHONE_CALL = 0x00000002
    const val KEYCODE_PHONE_END = 0x00000003
    const val KEYCODE_PHONE_END_MUTE = 0x00000004
    const val KEYCODE_HFP = 0x00000005
    const val KEYCODE_SELECTOR_NEXT = 0x00000006
    const val KEYCODE_SELECTOR_PREVIOUS = 0x00000007
    const val KEYCODE_SETTING = 0x00000008
    const val KEYCODE_MEDIA = 0x00000009
    const val KEYCODE_RADIO = 0x0000000A
    const val KEYCODE_NAV = 0x0000000B
    const val KEYCODE_SRC = 0x0000000C
    const val KEYCODE_MODE = 0x0000000D
    const val KEYCODE_BACK = 0x0000000E
    const val KEYCODE_SEEK_SUB = 0x0000000F
    const val KEYCODE_SEEK_ADD = 0x00000010
    const val KEYCODE_VOLUME_SUB = 0x00000011
    const val KEYCODE_VOLUME_ADD = 0x00000012
    const val KEYCODE_MUTE = 0x00000013
    const val KEYCODE_OK = 0x00000014
    const val KEYCODE_MOVE_LEFT = 0x00000015
    const val KEYCODE_MOVE_RIGHT = 0x00000016
    const val KEYCODE_MOVE_UP = 0x00000017
    const val KEYCODE_MOVE_DOWN = 0x00000018
    const val KEYCODE_MOVE_UP_LEFT = 0x00000019
    const val KEYCODE_MOVE_UP_RIGHT = 0x0000001A
    const val KEYCODE_MOVE_DOWN_LEFT = 0x0000001B
    const val KEYCODE_MOVE_DOWN_RIGHT = 0x0000001C
    const val KEYCODE_TEL = 0x0000001D
    const val KEYCODE_MAIN = 0x0000001E
    const val KEYCODE_MEDIA_START = 0x0000001F
    const val KEYCODE_MEDIA_STOP = 0x00000020
    const val KEYCODE_VR_START = 0x00000021
    const val KEYCODE_VR_STOP = 0x00000022
    const val KEYCODE_NUMBER_0 = 0x00000023
    const val KEYCODE_NUMBER_1 = 0x00000024
    const val KEYCODE_NUMBER_2 = 0x00000025
    const val KEYCODE_NUMBER_3 = 0x00000026
    const val KEYCODE_NUMBER_4 = 0x00000027
    const val KEYCODE_NUMBER_5 = 0x00000028
    const val KEYCODE_NUMBER_6 = 0x00000029
    const val KEYCODE_NUMBER_7 = 0x0000002A
    const val KEYCODE_NUMBER_8 = 0x0000002B
```

```
    const val KEYCODE_NUMBER_9 = 0x0000002C
    const val KEYCODE_NUMBER_STAR = 0x0000002D

    const val KEYCODE_NUMBER_POUND = 0x0000002E

    const val KEYCODE_NUMBER_DEL = 0x0000002F
    const val KEYCODE_NUMBER_CLEAR = 0x00000030
    const val KEYCODE_NUMBER_ADD = 0x00000031

    const val MSG_TOUCH_PAD_DOWN = 0x0001005A
    const val MSG_TOUCH_PAD_MOVE = 0x0001005B
    const val MSG_TOUCH_PAD_UP = 0x0001005C
    const val MSG_TOUCH_PAD_PINCH = 0x0001005D
    const val MSG_CMD_FOCUS_CHANGE = 0x0001005E

    // Data通道相关消息 (UPDATE)
    const val MSG_DATA_MD_TRANSFER_START = 0x00070007
    const val MSG_DATA_MD_TRANSFER_SEND = 0x00070008
    const val MSG_DATA_MD_TRANSFER_END = 0x00070009
    const val MSG_DATA_HU_UPDATE_START = 0x0007800A
    const val MSG_DATA_HU_UPDATE_END = 0x0007800B

    // 无线连接，蓝牙通道相关消息
    const val MSG_WIRELESS_INFO_REQUEST = 0x00100001
    const val MSG_WIRELESS_INFO_RESPONSE = 0x00108002
    const val MSG_WIRELESS_TARGET_INFO_REQUEST = 0x00100004
    const val MSG_WIRELESS_TARGET_INFO_RESPONSE = 0x00108005
    const val MSG_WIRELESS_REQUEST_IP = 0x00108006
    const val MSG_WIRELESS_RESPONSE_IP = 0x00100007
    const val MSG_WIRELESS_MD_STATUS = 0x00100008
    const val MSG_WIRELESS_HU_STATUS = 0x00108009
    const val MSG_WIRELESS_BT_AUDIO_INFO = 0x00108010
    const val MSG_WIRELESS_ENABLE_BT_AUDIO = 0x00108011
}
```