

Grandstream Networks, Inc.

Grandstream GXV3450/GXV3470/GXV3480

SDK Framework Service Guide



COPYRIGHT

©2022 Grandstream Networks, Inc. <http://www.grandstream.com>

All rights reserved. Information in this document is subject to change without notice. Reproduction or transmittal of the entire or any part, in any form or by any means, electronic or print, for any purpose without the express written permission of Grandstream Networks, Inc. is not permitted.

The latest electronic version of this guide is available for download here:

<http://www.grandstream.com/support>

Grandstream is a registered trademark and Grandstream logo is trademark of Grandstream Networks, Inc. in the United States, Europe and other countries.

CAUTION

Changes or modifications to this product not expressly approved by Grandstream, or operation of this product in any way other than as detailed by this guide, could void your manufacturer warranty.

WARNING

Please do not use a different power adapter with devices as it may cause damage to the products and void the manufacturer warranty.



Table of Content

SUPPORTED DEVICES.....	7
CHANGE LOG.....	8
SDK Framework Version 1.2.0.....	8
SDK Framework Version 1.2.1.....	8
OVERVIEW.....	9
APPLICATION BUILDING.....	10
Create dir “gslibs” under App Module.....	10
Build.gradle.....	10
Main API list in JARs.....	11
USE APIS.....	13
Initialize ApiClient.....	13
BaseAccountApi.....	14
<i>Account Information.....</i>	<i>14</i>
<i>Monitor Account Status.....</i>	<i>14</i>
Start Account Status Monitor.....	14
Stop Account Status Monitor.....	14
Monitor Account using AccountStatusListener.....	15
<i>Update Account.....</i>	<i>15</i>
BaseCallApi.....	15
<i>Line Id.....</i>	<i>15</i>
<i>Make a Call.....</i>	<i>16</i>
<i>End a Call.....</i>	<i>17</i>
<i>Monitor Incoming Call.....</i>	<i>17</i>
Start Call Status Monitor.....	17
Stop Call Status Monitor.....	18
Monitor Call Status using CallStatusListener.....	18
<i>Monitor Handset Hook ON/OFF.....</i>	<i>18</i>
Start Handset Hook Event Monitor.....	18



Stop Handset Hook Event Monitor.....	19
CallStatusListener Implement.....	19
<i>Transfer a Call</i>	19
Default Phone App.....	20
<i>Implement Customized Phone Service</i>	20
<i>Register Customized Phone Service</i>	21
<i>Configure the Default Phone App</i>	21
SmsManagerApi.....	22
<i>Send an SMS</i>	22
<i>Resend Failed SMS</i>	23
<i>Receive SMS</i>	23
<i>Delete SMS</i>	24
AudioRouteApi.....	25
<i>Switch Audio Route</i>	25
<i>Check Audio Route</i>	25
<i>EHS Headset</i>	26
<i>Always Ring Speaker</i>	26
CallSettingApi.....	27
DeviceApi.....	27
CallToneApi.....	27
Gs Core SDK.....	30
<i>Device Manager</i>	30
Get device connected state.....	30
Reboot device.....	30
DND Manager.....	30
Get Device MAC.....	31
CONTACTS.....	32
Changes for ContactsContract.Data.....	32
Changes for ContactsContract.RawContacts.....	32
Changes for ContactsContract.Contacts#CONTENT_FILTER_URI.....	33
Create a Contact.....	33



Update a Contact.....	34
Read Contacts.....	35
Delete a Contact.....	35
CALL LOGS.....	37
Changes for CallLog.Calls.....	37
Create a Call Log.....	38
Read Call Logs.....	38
Update Call Logs.....	39
Delete call logs.....	39
LED.....	41
GsLightsManager.....	42
PROGRAMMABLE KEYS.....	43
Monitor Extension Module Click Event.....	43
<i>Start Ext Click Event Monitor.....</i>	<i>43</i>
<i>Stop Ext Click Event Monitor.....</i>	<i>43</i>
Monitor Ext Event Using MpkStatusListener.....	43
DEVELOP APPS WITH ADB.....	44
Enable Developer Mode on Device.....	44
ADB Connect.....	44
Allow Debug on Device.....	44
Check ADB Connect Status.....	44
ADB Disconnect.....	44
API DEMO.....	45



Table of Figures

Figure 1 : Create gslibs under App Module.....	10
Figure 2 : Default Phone app.....	22

Table of Tables

Table 1 : Supported Devices.....	7
Table 2 : Descriptions for JARs.....	11
Table 3 : Descriptions for APIs.....	11
Table 4 : ContactsContract.Data.....	32
Table 5 : ContactsContract.RawContacts.....	32
Table 6 : CallLog.Calls.....	37
Table 7 : LED Colors and Frequencies.....	41



SUPPORTED DEVICES

The following table shows Grandstream products supporting SDK APIs listed on this guide:

Table 1: Supported Devices

Model	Supported	Firmware
GXV3470	Yes	1.0.1.1 or Higher
GXV3480	Yes	1.0.1.1or Higher
GXV3450	Yes	1.0.1.1 or Higher



CHANGE LOG

This section documents significant changes from previous versions of SDK Framework Service guide for GXV3450/GXV3470/GXV3480. Only major new features or major document updates are listed here. Minor updates for corrections or editing are not documented here.

SDK Framework Version 1.2.0

- Release SDK for GXV3450-1.0.1.1, GXV3480-1.0.1.1, GXV3470-1.0.0.4.
- the SDK inheritance from GXV33X0 SDK.

SDK Framework Version 1.2.1

- Open new API BaseCallApi.confAddLine, BaseCallApi.confAddLines for conerence.
- Add introduce about Line Id in docs.



OVERVIEW

GXV3450/GXV3470/GXV3480 operating system is developed based on Android™ platform. Besides inheriting the Android interface functions, more interfaces have been supported from users requirements. This document describes how to use the APIs for users' application development on GXV3450/GXV3470/GXV3480.



Note:

Before starting the API demo or testing your own apps, please upgrade your GXV3470/GXV3480 /GXV3450 to the latest firmware version. The firmware release information can be found in the following link: <http://www.grandstream.com/support/firmware>



APPLICATION BUILDING

With GS JAR files, users can start building the apps with Android Studio (or other IDE) following the steps on this guide.

Create dir “gslibs” under App Module

Under App module, create “gslibs” directory. Then copy the JAR files “gsapi-1.x.x.jar” under “gslibs” directory so the JAR files are reachable from the app files.

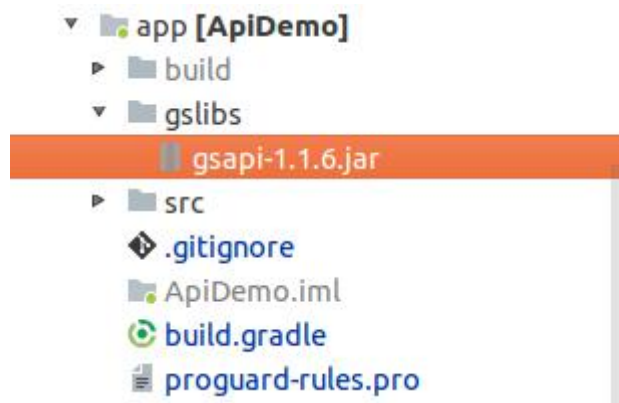


Figure 1: Create gslibs under App Module

Build.gradle

Add dependencies on the build.gradle file using [compileOnly](#) or [provided](#). Please see JAR files path below for reference:

```
dependencies {
    . . . . .
    compileOnly files ('gslibs/gsapi-1.2.0.jar')
}
```

Note:

Please do not package the JARs into APK.

Please refer to below table that has listed the JARs that can be included when compiling your app. They should be added in “build.gradle” file if used.

Table 2: Descriptions for JARs

Package	Description
com.gs.common	Support Gs ApiClient
com.gs.account	Support Gs Account APIs
com.gs.phone	Support Gs Phone APIs
com.gs.contacts	Support Gs Contacts APIs
com.gs.callog	Support Gs Callog APIs
com.gs.sms	Support Gs sms APIs

Main API list in JARs

Here is a list of main APIs included in the above JARs:

Table 3: Descriptions for APIs

API	Description
AudioRouteApi	APIs related to audio route. This is used for switching audio route and checking audio route.
BaseAccountApi	APIs related to accounts. This is used for calls.
BaseLineApi	APIs related to lines. This is used for creating line channel upon outgoing call and incoming call.
BaseCallApi	APIs related to calls. This is used for making a call, receiving a call and etc.
BasePhoneService	APIs for the target app as default phone app.
ContactsContext	Useful API constants for Create-Read-Update-Delete (CRUD) contacts.
CallogContext	Useful API constants for Create-Read-Update-Delete (CRUD) call logs.
SmsManagerAPI	APIs related to SMS. This is used for sending SMS, adding SMS lis and etc.
CallToneApi	Useful API constants for playing the call tones, such as incoming call ringtone, call waiting tone, busy tone, dial tone, DTMF tone, confirmation tone, auto-answer tone, DND tone, reorder tone, etc.



Note:

All classes, functions and members of classes are NOT part of any supported API, which are not defined in



gsApiExternal-en. If the code written depends on these, please proceed at your own risks. This code and its internal interfaces are subject to change or deletion without notice.



USE APIs

All APIs are named as **XXXApi**, such as: **BaseAccountApi**, **BaseCallApi**, **BaseLineApi**, **SmsManagerApi**.

- Before using the APIs, users must initialize ApiClient first
- Use a function such as XXXApi.function1()
- Use a monitor such as XXXApi.function2(callback2)

Details of the API functions can be found under **gsApiExternal-en** directory.

Initialize ApiClient

Supported Products: GXV34X0.

App can use ApiClient by com.gs.com. To initialize ApiClient, use [setContext](#), [addApi](#) and [build](#) functions in the Application block.



Note:

Initialization shall be performed only one time for each process. Please do not initialize the same process more than once.

Please follow below code to initialize ApiClient:

```
public class DemoApplication extends Application {
    @Override
    public void onCreate() {
        super.onCreate();

        ApiClient.builder().setContext(getApplicationContext())
            .addApi(BaseAccountApi.API)
            .addApi(BaseCallApi.API)
            .addApi(BaseLineApi.API)
            .addApi(SmsManagerApi.API)
            .addApi(AudioRouteApi.API)
            .build();
    }
}
```



```
}  
}
```

BaseAccountApi

Supported Products: GXV34X0.

Account API can be used to obtain account information, modify account information and monitor account changes.

Account Information

You can get all account information by using **BaseAccountApi.getAllSipAccounts**. Here is an example on how to use Account Info APIs:

```
public void getAllAccount() {  
    List<SipAccount> sipAccounts = BaseAccountApi.getAllSipAccounts();  
}
```

Monitor Account Status

Users can monitor account status by using **BaseAccountApi.addStatusListener** and **BaseAccountApi.removeStatusListener**. Here is an example on how to use monitor account status APIs.

Start Account Status Monitor

```
private void startMonitorAccountChange() {  
    mAccountStatusListener = new AccountStatusListener();  
    BaseAccountApi.addStatusListener("MonitorAccount",  
        mAccountStatusListener.callback,  
        AccountContext.ListenType.SIP_ACCOUNT_STATUS, false);  
}
```

Stop Account Status Monitor

```
private void stopMonitorAccountChange() {  
    BaseAccountApi.removeStatusListener(mAccountStatusListener.callback);  
    mAccountStatusListener.destroy();  
    mAccountStatusListener = null;  
}
```



Monitor Account using AccountStatusListener

```
private class MyAccountStatusListener extends AccountStatusListener {
    @Override
    public void onSipAccountStatusChanged(List<SipAccount> list, SipAccount
sipAccount) {
        }
};

AccountStatusListener mAccountStatusListener = null;
```

Update Account

Account information can be updated using **BaseAccountApi.updateSipAccount**. Here is an example showing how to use Update Account function:

```
private void updateAccount() {
    BaseAccount account = BaseAccountApi.getAccountById(0);
    Log.d(TAG, "updateAccount,original:"+account);
    if(account != null && account instanceof SipAccount){
        account.setAccountName("Test Account");
        boolean ret = BaseAccountApi.updateSipAccount((SipAccount) account);
        if(ret){
            Log.d(TAG, "updateAccount,changed to:"+account);
        }
    }
}
```

BaseCallApi

Call functions allow users to make calls, end calls and monitor call status.

Line Id

Before using the BaseCallAPI, you need to understand the concept of Line Id. Each call generates a Line Id for identification and management. The Line Id is generated by the system when a line is created. The application can obtain the line information by listening the line status changes. The line information contains the Line Id. When making a call, you can also obtain the line parameters by listening to the line status changes.



Make a Call

Supported Products: GXV34X0.

Users can make a call by using ***BaseCallApi.makeCalls***. Here is an example on how to use Call function API to make a call:



```
public void call(String num) {
    List<DialingInfo> dialingInfos = new ArrayList<DialingInfo>();
    DialingInfo dialingInfo = new DialingInfo();
    dialingInfo.setAccountId(BaseAccountApi.getDefaultAccount().getAccountId());
    dialingInfo.setOriginNumber(num);
    dialingInfos.add(dialingInfo);
    DialResults dialResults = BaseCallApi.makeCalls(dialingInfos);
    int result = dialResults.getDialResult(dialingInfo);
}
```

End a Call

Supported Products: GXV34X0.

User can end a call by using **BaseCallApi.endCall**. Here is an example on how to use Call function API to end a call:

```
public void endCall(View view) {
    DemoApplication app = (DemoApplication) getApplication();
    // the line id could be saved by monitor call line status change
    int lineId = app.getCurLineId();
    BaseCallApi.endCall(lineId);
}
```

Monitor Incoming Call

Supported Products: GXV34X0.

In order to answer incoming call, users need monitor the incoming call first. Here is an example of monitoring an incoming call.

Start Call Status Monitor

To start monitoring a line's status, use **BaseCallApi.addStatusListener** with **LINE_STATUS**, and monitor a call line which has id **LINE_ID**.

```
private void startMonitorCallLines() {
    mCallStatusListener = new MyCallStatusListener();
    BaseCallApi.addStatusListener("MonitorCall",
        mCallStatusListener.callback,
        PhoneContext.ListenType.LINE_ID |
        PhoneContext.ListenType.LINE_STATUS,
        false);
}
```



```
}
```

Stop Call Status Monitor

```
private void stopMonitorCallLines() {
    BaseCallApi.removeStatusListener(mCallStatusListener.callback);
    mCallStatusListener.destroy();
    mCallStatusListener = null;
}
```

Monitor Call Status using CallStatusListener

```
private class MyCallStatusListener extends CallStatusListener {
    @Override
    public void onLineStatusChanged(int notifyType, BaseLine baseLine,
    List<BaseLine> list) {
    }

    @Override
    public void onLineIdChanged(int oldLineId, int newLineId) {
    }
}

private CallStatusListener mCallStatusListener = null;
```

Monitor Handset Hook ON/OFF

Supported Products: GXV34X0.

Similar to monitoring incoming call, handset monitoring uses ***add/removeStatusListener***. Here is an example of monitoring the hook event status:

Start Handset Hook Event Monitor

Users can monitor handset hook event status by using ***BaseCallApi.addStatusListener*** with ***HOOK_EVENT_STATUS***.

```
private void startMonitorHandsetChange() {
    mHookStatusListener = new MyHookStatusListener();
    BaseCallApi.addStatusListener("MonitorHandset",
        mHookStatusListener.callback,
        PhoneContext.ListenType.HOOK_EVENT_STATUS, false);
}
```



```
}
```

Stop Handset Hook Event Monitor

```
private void stopMonitorHandsetChange() {
    BaseCallApi.removeStatusListener(mHookStatusListener.callback);
    mHookStatusListener.destroy();
    mHookStatusListener = null;
}
```

CallStatusListener Implement

```
private class MyHookStatusListener extends CallStatusListener {
    @Override
    public void onHookEventChanged(int device, boolean isOffHook) {
    }
}

private CallStatusListener mHookStatusListener = null;
```

Transfer a Call

Supported Products: GXV34X0.

Users can perform blind transfer and attended transfer through ***transferBlind/transferAttended***. The current call needs to be held before transfer.

After blind transfer, the current call ends directly, and the transferred party directly calls the transfer target. After attended transfer, the current call will not be ended after the specified transfer, and the phone will call the transfer target. When the transfer target does not answer the call, the transfer can be cancelled/completed immediately; the phone can transfer immediately or split after the transfer target answers.

The relevant interfaces are as follows:

```
BaseCallApi.transferBlind(int lineId, String number);
BaseCallApi.transferAttended(int lineId, String number);
BaseCallApi.transferAttendedCancel(int lineId);
BaseCallApi.endCall(int lineId);
BaseCallApi.transferAttendedEnd(int lineId);
BaseCallApi.transferSplit();
```



Default Phone App

Supported Products: GXV34X0.

Multiple phone applications are allowed in Grandstream phone system, and all of them can take control of calls using **BaseCallApi** functions. However, only one phone application can respond to line events and on/off-hook events, which is the default phone application. A default phone application is required to handle the logic of a call.

In general, **System Phone** is the default phone application and it can meet most needs. However, users can also develop your own phone application and make it the default phone APP.

Implement Customized Phone Service

Customizing your own phone service requires extending **BasePhoneService** and handling onhook/offhook events.



Gs JARs currently do not support subscription for phone line change events when **System Phone** is not the default phone app. This means once a customized app is chosen to be the default phone app, it's not allowed to use **System Phone** line for making calls anymore.

```
public class DemoService extends BasePhoneService {
    @Override
    public void onHookEvent(boolean offHook) {
        super.onHookEvent(offHook);
        //TODO:do something
    }

    public void onEHSHookEvent(boolean offHook) {
        super.onEHSHookEvent(offHook);
        //TODO:do something
    }

    public void onLineStateChanged(int lineId, int status) {
        super.onLineStateChanged(lineId, status);
        //TODO:do something
    }

    /**
```



```

    * return true means this click event was cost by this service.
    * else this event will call the system emergency dialer.
    */
    public boolean onEmergencyCallButtonClicked(int lineId, int status) {
        super.onEmergencyCallButtonClicked(lineId, status);
        //TODO:do something
        return true;
    }

    /**
     * This method will be called when clicking "Touch to return to call" in
     the top status bar, and you can start the call interface here.
     */
    @Override
    public void onShowCallingView() {
        super.onShowCallingView();
    }
}

```

Register Customized Phone Service

```

<service android:name=".DemoService" >

    <intent-filter>

        <action android:name="com.gs.phone.service" />

    </intent-filter>

</service>

```

Configure

the Default Phone App

1. Install the target phone app.
2. Open System Settings and go under APPs → Default Application → Phone app.
3. Choose the target phone app as the default phone app.



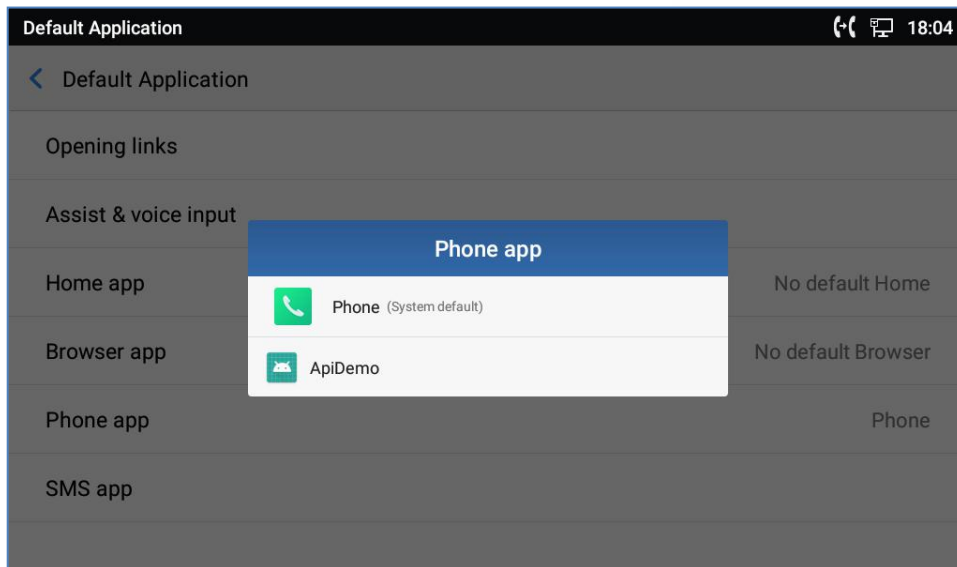


Figure 2: Default Phone app

SmsManagerApi

Supported Products: GXV34X0

SMS functions allow users to send/receive SMS as well as handle failed SMS.

Send an SMS

Use **SmsManagerApi.sendSmsAfterSaved** to send an SMS to a single contact or a group of contacts.

```
int accountId = 0;

String numberStr = "36324";

String content = "Test SMS to single one.";

long msgId = SmsManagerApi.sendSmsAfterSaved(accountId, numberStr, content);

if(msgId > -1){

    Log.d(TAG, "Sending sms (" + msgId + ") ..");

}

String group = numberStr + "_," + "36325";

content = "Test SMS to a group of contacts.";
```



```
msgId = SmsManagerApi.sendSmsAfterSaved(accountId, group, numberStr, content);

if(msgId > -1){

    Log.d(TAG, "Sending sms (" + msgId + ")..");

}
```

The sending result can be monitored using **SmsManagerApi.addSmsListener**.

```
SmsManagerApi.addSmsListener(new SmsListener() {

    @Override
    public void onSend(long msgId, boolean sendOk) {
        Log.d(TAG, "This is main thread.");
        Log.d(TAG, "sms (" + msgId + ") is send " + (sendOk ? "success" : "fail"));
    }

});
```

Resend Failed SMS

Use **SmsManagerApi.resendFailedSms** to resend a failed SMS.

```
boolean ret = SmsManagerApi.resendFailedSms(msgId);

if(ret){

    Log.d(TAG, "ReSending sms (" + msgId + ")..");

}else{

    Log.e(TAG, "ReSending sms (" + msgId + ") fail.");

}
```

Receive SMS

Users can register a **BroadcastReceiver** with an **SmsContext.SMS_RECEIVED_ACTION** to receive SMS and use the constants defined in **SmsContext.ReceiveSms** to get the content of the received SMS.

```
private SmsReceiver smsReceiver;

private void registerSmsReceiver(){

    IntentFilter filter = new IntentFilter();
```



```

    filter.addAction(SmsContext.SMS_RECEIVED_ACTION);

    smsReceiver = new SmsReceiver();

    registerReceiver(smsReceiver, filter);
}

private void unRegisterSmsReceiver() {

    if(smsReceiver != null) {

        unregisterReceiver(smsReceiver);

    }

}

class SmsReceiver extends BroadcastReceiver{

    @Override

    public void onReceive(Context context, Intent intent) {

        if(SmsContext.SMS_RECEIVED_ACTION.equals(intent.getAction())) {

            long id = intent.getLongExtra(SmsContext.ReceiveSms.ID, -1);

            String number =

intent.getStringExtra(SmsContext.ReceiveSms.NUMBER);

            int accountId =

intent.getIntExtra(SmsContext.ReceiveSms.ACCOUNT_ID, -1);

            String content =

intent.getStringExtra(SmsContext.ReceiveSms.CONTENT);

        }

    }

}

```

Delete SMS

Use **SmsManagerApi.removeSmsById** and **SmsManagerApi.removeSmsByType** to delete a SMS. The interfaces are defined as follows:



```
public static int removeSmsById(long smsId);
```

Delete SMS based on smsId;

Parameter: smsId, SMS ID;

Return Value: int <0:error; ≥0: the number of deleted sms

Return 0: there is no such message in the database; Return ≥0: delete successfully.

```
public static int removeSmsByType(int removeType);
```

Delete SMS based on removeType;

Parameter removeType: 0 - delete received messages, 1 - delete sent messages, 2 - delete draft messages;

Return Value: int <0: error; ≥0: the number of deleted sms;

Return 0: there is no such message in the database; Return ≥0: delete successfully.

AudioRouteApi

Supported Products: GXV34X0

With AudioRouteApi, the audio route can be switched or checked by the app.

Switch Audio Route

Use **AudioRouteApi.switchVoiceToXXX** to switch audio route. Please make sure the route switched to is supported by the product. Otherwise, it will fail to switch.

```
AudioRouteApi.switchVoiceToSpeaker();  
AudioRouteApi.switchVoiceToHandset();  
AudioRouteApi.switchVoiceToRJ9Headset();  
AudioRouteApi.switchVoiceToBlueToothHeadset();  
AudioRouteApi.switchVoiceToHdmi();  
AudioRouteApi.switchVoiceToEarphone();  
AudioRouteApi.switchVoiceToUsbHeadset();
```

Check Audio Route

Use **AudioRouteApi.isVoiceOnXXX** to check the current audio route.

```
AudioRouteApi.isVoiceOnSpeaker();  
AudioRouteApi.isVoiceOnHandset();  
AudioRouteApi.isVoiceOnRJ9Headset();  
AudioRouteApi.isVoiceOnBlueToothHeadset();
```



```
AudioRouteApi.isVoiceOnHdmi();
AudioRouteApi.isVoiceOnEarphone();
AudioRouteApi.isVoiceOnUsbHeadset();
```

EHS Headset

- **setEhsHookStatus**: to set the EHS status.
- **isEhsOffHook**: to check whether the EHS headset is off hook.
- **isEhsHeadsetConnected**: check whether the EHS headset connected or not.



1. When the EHS headset is onhook, EHS will not play any sound.
 2. When calling **setEhsHookStatus**, the audio route must be on EHS headset, which means the result of **isVoiceOnRJ9Headset()** must be true. Also, please do not switch to other audio route within 500ms after calling **setEhsHookStatus**.
-

Always Ring Speaker

If the config WebUI → Phone Settings → Call Settings → “Always Ring Speaker” is checked, this means an incoming call will ring with speaker, even though current voice channel is not on speaker.

In order to perform as above, the custom phone app should use the api **switchCurrentVoiceWithSpeaker** and **switchCurrentVoiceWithoutSpeaker** when the ringing finished.

Below is a sample code when a new call is incoming.

```
private class MyCallStatusListener extends CallStatusListener {
    @Override
    public void onLineStatusChanged(int notifyType, BaseLine baseLine,
    List<BaseLine> list) {
        final BaseLine line = baseLine;
        if (line.getStatus() == 2) { // 2 means ringing
            if (CallSettingApi.isAlwaysRingSpeaker()) {
                AudioRouteApi.switchCurrentVoiceWithSpeaker();
                startTone(); // start ringing
            }
        } else {
            if (mPrevStatus == 2) {
```



```
        stopTone(); // stop ringing
        AudioRouteApi.switchCurrentVoiceWithoutSpeaker();
    }
}
mPrevStatus = line.getStatus();
}
}
```

CallSettingApi

Supported Products: GXV34X0.

With **CallSettingApi**, the config status about call can be checked by the app.

DeviceApi

Supported Products: GXV34X0.

DeviceApi provides the management and information SDK about the device.

Below is a sample code to get the system information.

```
SystemInfo systemInfo = DeviceApi.getSystemInfo();
String productBaseName = systemInfo.getProductBase();
String systemVersion = systemInfo.getSystemVersion();
```

CallToneApi

Supported Products: GXV34X0.

Below are the related CallToneApi interfaces:

```
//Check if the tone is playing
public static boolean isAnyToneInPlay();

//Check if the ringtone is playing
public static boolean isRingToneInPlay();

//Check if the dialtone is playing
public static boolean isDialToneInPlay();

//Check if the dtmf tone is playing
```



```
public static boolean isDtmfToneInPlay();

//Stop all tone
public static void stopAllTone();

//Stop all ringtone
public static void stopRingtone();

//Start play dtmf tone
public static void startDtmfTone(int keyValue);

//Stop play dtmf tone
public static void stopDtmfTone(int keyValue);

//Start play call waiting tone
public static void startCallWaitingTone();

//Stop play call waiting tone
public static void stopCallWaitingTone();

//Start play ringback tone
public static void startRingbackTone();

//Stop play ringback tone
public static void stopRingbackTone();

//Start play busy tone
public static void startBusyTone();

//Stop play busy tone
public static void stopBusyTone();

//Start play dial tone
public static void startDialTone();

//Stop play dial tone
public static void stopDialTone();

//Start play second dial tone
public static void startSecondDialTone();
```



```
//Stop play second dial tone
public static void stopSecondDialTone();

//Start play confirm tone
public static void startConfirmTone();

//Stop play confirm tone
public static void stopConfirmTone();

//Start play reorder tone
public static void startReorderTone();

//Stop play reorder tone
public static void stopReorderTone();

//Play the matched ringtone according to the line
public static void startRingtone(int lineId);

//Start play system ringtone
public static void playSystemRingtone();

//Start play the ringtone in the DND mode
public static void playDndTone();

//Start play dial tone when there is unread voice mail
public static void startVMDialTone();

//Stop play dial tone when there is unread voice mail
public static void stopVMDialTone();

//Start play auto answer tone
public static void startAutoAnswerTone();

//Stop play auto answer tone
public static void stopAutoAnswerTone();

//Gets the type of tone currently playing
public static int getCurToneType();
```



Gs Core SDK

These SDK can be used without init ApiClient. These SDKs are named as **XXXManager**, such as: **GsDevStateManager**.

Device Manager

Supported Products: GXV34X0.

Get device connected state.

```
// is 3.5mm earphone device connected
GsDevStateManager.instance().isDeviceConnected(GsDevStateManager.NAME_EARPHONE)
;

// is ehs device connected
GsDevStateManager.instance().isDeviceConnected(GsDevStateManager.NAME_EHS);

// is hdmi device connected
GsDevStateManager.instance().isDeviceConnected(GsDevStateManager.NAME_HDMI);

// is hdmi in device connected
GsDevStateManager.instance().isDeviceConnected(GsDevStateManager.NAME_HDMI_IN);

// is usb headset device connected
GsDevStateManager.instance().isDeviceConnected(GsDevStateManager.NAME_USB_HEADSET);
```

Reboot device

```
GsDevStateManager.instance().reboot();
```

DND Manager

```
DndManager.instance().setDndOn();
DndManager.instance().setDndOff();
DndManager.instance().isDndOn();
```



Get Device MAC

```
GsDevStateManager.getInstance().getDeviceMac();
```



CONTACTS

Supported Products: GXV34X0.

android.content.ContentResolver has already provided Create-Read-Update-Delete (CRUD) APIs for operating contacts. Please refer to below links for the APIs information:

<https://developer.android.google.cn/reference/android/content/ContentProvider>

For more details of the contacts database, please refer to:

<https://developer.android.google.cn/reference/android/provider/ContactsContract>.

By using **android.content.ContentResolver** to **CRUD** contacts, it should be able to meet most of the needs.

IMPORTANT:

Please make sure to read below content for the changes applied to Grandstream supported products before you **CRUD** contacts. Some new columns and some reserved columns are used in the database table to store important information. These columns' keys and constant values are defined in **com.gs.contacts.context.ContactsContext**. Please refer to **gsApiExternal-en** for more details.

Changes for ContactsContract.Data

Here is the change in contacts database table ContactsContract.Data:

Table 4: ContactsContract.Data

Key	Type	Description
ContactsContext.ContactsItem#ITEM_ACCOUNT_ID	INTEGER (long)	Special use.

Changes for ContactsContract.RawContacts

Here is the change in contacts database table ContactsContract.RawContacts:

Table 5: ContactsContract.RawContacts

Key	Type	Description
ContactsContext.ContactsItem#ITEM_SORT_KEY_T9	TEXT	New column.
ContactsContext.ContactsItem#ITEM_SORT_KEY_T9_FL	TEXT	New column.



Changes for ContactsContract.Contacts#CONTENT_FILTER_URI

In addition to the Android native filter, a function for fuzzy searching is also added. With fuzzy searching, users can query by a condition such like “number=123” to get all matching results with 123 included in the number such as “123456” or “0123” instead of only “123”. This filter can be enabled by setting **ContactsContext.SearchSnippets.FUZZY_KEY** as follows:

```
Uri.Builder builder = ContactsContract.Contacts.CONTENT_FILTER_URI.buildUpon();

builder.appendPath(query);

builder.appendQueryParameter(ContactsContract.DIRECTORY_PARAM_KEY,

    String.valueOf(directoryId));

builder.appendQueryParameter(ContactsContract.SearchSnippets.

    DEFERRED_SNIPPETING_KEY, "1");

builder.appendQueryParameter(com.gs.contacts.context.ContactsContext.

    SearchSnippets.FUZZY_KEY, "1");

Uri uri = builder.build();

Cursor cursor = mContext.getContentResolver().query(uri, null, null, null,

    null);
```

Create a Contact

A contact can be added and all the parameters can be manipulated. Here is an example:

```
public void addContact(String name, String number, long accountId) {
    <ContentProviderOperation> ops
        = new ArrayList<ContentProviderOperation>();
    int rawContactInsertIndex = ops.size();
    ops.add(ContentProviderOperation
        .newInsert(ContactsContract.RawContacts.CONTENT_URI)
        .withValue(ContactsContract.RawContacts.ACCOUNT_TYPE, null)
        .withValue(ContactsContract.RawContacts.ACCOUNT_NAME, null)
        .withYieldAllowed(true).build());
    ops.add(ContentProviderOperation
        .newInsert(ContactsContract.Data.CONTENT_URI)
        .withValue(ContactsContext.ContactsItem.ITEM_ACCOUNT_ID, accountId)
        .withValueBackReference(ContactsContract.Data.RAW_CONTACT_ID,
```



```

        rawContactInsertIndex)
    .withValue (ContactsContract.Data.MIMETYPE,
        ContactsContract.CommonDataKinds.
        StructuredName.CONTENT_ITEM_TYPE)
    .withValue (ContactsContract.CommonDataKinds.StructuredName
        .DISPLAY_NAME, name)
    .withYieldAllowed(true).build());
ops.add(ContentProviderOperation
    .newInsert (android.provider.ContactsContract.Data.CONTENT_URI)
    .withValueBackReference (ContactsContract.CommonDataKinds.Phone
        .RAW_CONTACT_ID, rawContactInsertIndex)
    .withValue (ContactsContract.CommonDataKinds.Phone.MIMETYPE,
        ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE)
    .withValue (ContactsContract.CommonDataKinds.Phone.TYPE,
        ContactsContract.CommonDataKinds.Phone.TYPE_MOBILE)
    .withValue (ContactsContract.CommonDataKinds.Phone.NUMBER, number)
    .withYieldAllowed(true).build());

try {
    contentResolver.applyBatch (ContactsContract.AUTHORITY, ops);
} catch (Exception e) {
    e.printStackTrace();
}
}

```

Update a Contact

A contact can be added and all the parameters can be manipulated. Here is an example:

```

public void updateContact(long rawContactId, String name) {
    ArrayList<ContentProviderOperation> ops
        = new ArrayList<ContentProviderOperation>();
    ops.add(ContentProviderOperation
        .newUpdate (ContactsContract.Data.CONTENT_URI)
        .withSelection (ContactsContract.Data.RAW_CONTACT_ID+"=?",
            new String[] {rawContactId+""})
        .withValue (ContactsContract.CommonDataKinds.StructuredName
            .DISPLAY_NAME, name)
        .build());

    try {
        contentResolver.applyBatch (ContactsContract.AUTHORITY, ops);
    }
}

```



```

    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

Read Contacts

Here is an example to read contacts and manipulate all the parameters:

```

public void getContacts() {
    String[] projection = new String[]{
        ContactsContext.ContactsItem.ITEM_CONTACT_ID_IN_DATA,
        ContactsContext.ContactsItem.ITEM_ACCOUNT_ID,
        ContactsContext.ContactsItem.ITEM_PHONE_NUMBER,
        ContactsContext.ContactsItem.ITEM_DISPLAY_NAME
    };

    Cursor cursor = contentResolver
        .query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
            projection, null, null, null);

    if (cursor != null) {
        while (cursor.moveToNext()) {
            long contactId = cursor.getLong
                (cursor.getColumnIndex(ContactsContext.ContactsItem
                    .ITEM_CONTACT_ID_IN_DATA));

            long accountId = cursor.getInt(cursor.getColumnIndex
                (ContactsContext.ContactsItem.ITEM_ACCOUNT_ID));

            String number = cursor.getString(cursor.getColumnIndex
                (ContactsContext.ContactsItem.ITEM_PHONE_NUMBER));

            String name = cursor.getString(cursor.getColumnIndex
                (ContactsContext.ContactsItem.ITEM_DISPLAY_NAME));

        }
        cursor.close();
    }
}

```

Delete a Contact

The following is an example of deleting contacts:

```

public void deleteContactById(long contactId) {
    ArrayList<ContentProviderOperation> ops
        = new ArrayList<ContentProviderOperation>();
}

```



```
ops.add(ContentProviderOperation
    .newDelete(ContactsContract.RawContacts.CONTENT_URI)
    .withSelection(ContactsContract.RawContacts.CONTACT_ID
        + "=" + contactId, null)
    .build());
ops.add(ContentProviderOperation
    .newDelete(ContactsContract.Data.CONTENT_URI)
    .withSelection(ContactsContract.Data.CONTACT_ID
        + "=" + contactId, null)
    .build());
try {
    contentResolver.applyBatch(ContactsContract.AUTHORITY, ops);
} catch (Exception e) {
    e.printStackTrace();
}
}
```



CALL LOGS

Supported Products: GXV34X0.

android.content.ContentResolver has already provided Create-Read-Update-Delete(CRUD) APIs for operating calllogs. Please refer to below link for the APIs information:

<https://developer.android.google.cn/reference/android/content/ContentProvider>

For more details about calllog database, please refer to:

<https://developer.android.google.cn/reference/android/provider/CallLog>

In general, after calling the APIs in BaseCallApi, calllogs will be added or changed by default. Gs JARS do not provide additional APIs for independent calllog function. Using **android.content.ContentResolver** to **CRUD** calllogs should be able to meet most of the needs.

IMPORTANT:

Please make sure to read below content for the changes applied o Grandstream supported products before you **CRUD** calllogs. Some new columns and some reserved columns are used in the database table o store important information. These columns' keys and constant values are defined in **com.gs.calllog.context.CallLogContext**. Please refer to **gsApiExternal-en** for more details.

Changes for CallLog.Calls

The changes applied to CallLog.Calls are listed in below table:

Key	Type	Description
CallLogContext.CallLogItem#ITEM_ACCOUNT	INTEGER(long)	New column.
CallLogContext.CallLogItem#ITEM_CALL_MODE	INTEGER(int)	New column.
CallLogContext.CallLogItem#ITEM_MEDIA_TYPE	INTEGER(int)	New column.
CallLogContext.CallLogItem#ITEM_NUMBER_ORIGINAL	TEXT	New column.
CallLogContext.CallLogItem#ITEM_CONTACT_CACHE_NAME	TEXT	New column.
CallLogContext.CallLogItem#ITEM_END_TIME	TEXT	New column.
CallLogContext.CallLogItem#ITEM_IS_IN_CONFERENCE	INTEGER(int)	New column.
CallLogContext.CallLogItem#ITEM_GROUP_ID	INTEGER(long)	New column.

Table 6: CallLog.Calls



Note: Public static methods in CallLog.Calls are not recommended, you may get unexpected results.



Create a Call Log

As call log is closely related to call, we strongly suggest using **BaseCallApi** to make a call and save the call log by default. Here is an example:

```
public void addCalllog(String name,String number,long accountId) {
    ArrayList<ContentProviderOperation> ops
        = new ArrayList<ContentProviderOperation>();
    ops.add(ContentProviderOperation.newInsert(CallLogContext.CALL_LOG_URI)
        .withValue(CallLogContext.CallLogItem.ITEM_ACCOUNT, accountId)
        .withValue(CallLogContext.CallLogItem.ITEM_CACHE_NAME, name)
        .withValue(CallLogContext.CallLogItem.ITEM_CALL_MODE,
            PhoneContext.CallMode.SIP_CALL)
        .withValue(CallLogContext.CallLogItem.ITEM_START_TIME,
            System.currentTimeMillis())
        .withValue(CallLogContext.CallLogItem.ITEM_IS_IN_CONFERERENCE, false)
        .withValue(CallLogContext.CallLogItem.ITEM_NUMBER_ORIGINAL, number)
        .withValue(CallLogContext.CallLogItem.ITEM_NUMBER, number)
        .withValue(CallLogContext.CallLogItem.ITEM_CALL_TYPE,
            CallLogContext.CallLogType.TYPE_MISSED)
        .withValue(CallLogContext.CallLogItem.ITEM_DURATION, 0)
        .withValue(CallLogContext.CallLogItem.ITEM_MEDIA_TYPE,
            CallLogContext.MediaType.AUDIO)
        .build());
    try {
        contentResolver.applyBatch(CallLog.AUTHORITY, ops);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Read Call Logs

The following is an example of reading call logs using **ContentResolver**:

```
public void getCallLogs() {
    Cursor cursor = contentResolver.query(CallLogContext.CALL_LOG_URI,
        null, null, null, null);
    StringBuilder res = new StringBuilder("");
}
```



```

if(cursor!= null){
    if(cursor.moveToFirst()){
        do{
            long calllogId = cursor.getLong(
                cursor.getColumnIndex(CallLogContext.CallLogItem.ITEM_ID));
            String cacheName = cursor.getString(cursor.
                getColumnIndex(CallLogContext.CallLogItem.ITEM_CACHE_NAME));
            int mediaType = cursor.getInt(cursor.
                getColumnIndex(CallLogContext.CallLogItem.ITEM_MEDIA_TYPE));
        }while (cursor.moveToNext());
    }
    cursor.close();
}
}

```

Update Call Logs

Here is an example to update call logs using **ContentResolver**:

```

public void updateCalllogById(long callId) {
    ArrayList<ContentProviderOperation> ops
        = new ArrayList<ContentProviderOperation>();
    ops.add(ContentProviderOperation
        .newUpdate(CallLogContext.CALL_LOG_URI)
        .withSelection(CallLogContext.CallLogItem.ITEM_ID+"="+callId, null)
        .withValue(CallLogContext.CallLogItem.ITEM_DURATION,1000)
        .build());

    try {
        contentResolver.applyBatch(CallLog.AUTHORITY, ops);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

Delete call logs

Here is an example to delete call logs using **ContentResolver**:

```

public void deleteCalllogById(long callId) {
    ArrayList<ContentProviderOperation> ops
        = new ArrayList<ContentProviderOperation>();

```



```
ops.add(ContentProviderOperation
    .newDelete(CallLogContext.CALL_LOG_URI)
    .withSelection(CallLogContext.CallLogItem.ITEM_ID+"="+callId, null)
    .build());

try {
    contentResolver.applyBatch(CallLog.AUTHORITY, ops);
} catch (Exception e) {
    e.printStackTrace();
}
}
```



LED

Supported Products: GXV34X0.

In the Android standard interface, notification is the only method is use LED. Here is an example to use notification with LED:

```
int color = 0xFF00FF00;
int onMs = 1000;
int offMs = 1000;
NotificationManager manager = (NotificationManager)
getSystemService(NOTIFICATION_SERVICE);
Notification notification = null;
notification = new Notification.Builder(this)
    .setContentTitle("This is content title")
    .setContentText("This is content text")
    .setWhen(System.currentTimeMillis())
    .setSmallIcon(R.mipmap.ic_launcher)
    .setLargeIcon(BitmapFactory.decodeResource(getResources(),
R.mipmap.ic_launcher))
    .setLights(color, onMs, offMs)
    .build();

manager.notify(1, notification);
```

Please find a list of supported colors and frequencies for the LED on GXV3370, GXV3380 and GXV3350. If the color or the frequency setting is not supported by the device, the device will automatically choose the default color or the nearest frequency.

Table 7: LED Colors and Frequencies

Product	GXV3370	GXV3380	GXV3350
Red LED	On/Off	On/Off	On/Off
Green LED	On/Off	On/Off	On/Off
Blue LED	N/A	N/A	N/A
Keep on frequency	onMs: 1000, offMs: 0	onMs: 1000, offMs: 0	onMs: 1000, offMs: 0
Slow splash frequency	onMs: 1000, offMs: 3000	onMs: 1000, offMs: 3000	onMs: 1000, offMs: 3000
Fast splash frequency	onMs: 1000, offMs: 1000	onMs: 1000, offMs: 1000	onMs: 1000, offMs: 1000



GsLightsManager

Except controlling LED with Notification, the GsLightsManager provided functions to control the LED freely.

The following are method definition:

```
public void openCustom(int color, int shineType);
```

```
public void closeCustom();
```

The parameters:

shineType: control the shine type with fast splash, slow splash or keep on.

color: the LED color to open. According to the product, it can support red, green or blue.



Programmable Keys

Supported Products: GXV3450

Programmable Keys API can be used to monitor extension module click events, etc.

Monitor Extension Module Click Event

Users can monitor extension module click event by using ***MpkApi.addStatusListener*** and ***MpkApi.removeStatusListener***.

Here is an example on how to use monitor extension module click event APIs.

Start Ext Click Event Monitor

```
private void initExtClickListener() {  
    mMpKStatusListener = new MyMpKStatusListener();  
    MpkApi.addStatusListener("ExtEvent",  
        mMpKStatusListener.callback,  
        MpkContext.ListenType.EXT_EVENT);  
}
```

Stop Ext Click Event Monitor

```
private void removeExtClickListener() {  
    MpkApi.removeStatusListener(mMpKStatusListener.callback);  
    mMpKStatusListener.destroy();  
    mMpKStatusListener = null;  
}
```

Monitor Ext Event Using MpKStatusListener

```
private class MyMpKStatusListener extends MpKStatusListener {  
    @Override  
    public void onExtItemClick(MpkEntity entity) {  
    }  
};  
  
MpKStatusListener mMpKStatusListener = null;
```



DEVELOP APPS WITH ADB

Developing apps on Grandstream android devices requires the device to be connected to network. This is different from development using USB connection to the Android device directly.

Enable Developer Mode on Device

Please enable developer mode under device web UI. Log in device web UI and go to Settings→System security→Developer mode.

ADB Connect

Use the command “adb connect IP” to connect to the device. For example:

adb connect 192.168.100.1

Allow Debug on Device

After successful “adb connect”, the device will show a prompt requesting to allow debugging. Please choose “OK” to proceed. It is suggested to check “Always allow from this computer” so that the dialog will not show again when the device is connected to the same computer next time.

Check ADB Connect Status

Use the command “adb devices” to check the adb connect status. If below content is displayed, it means connection is successful:

List of devices attached

192.168.100.1:5555 device

ADB Disconnect

Use the command “adb disconnect” to disconnect all devices.



Only one PC is allowed to connect to the device at a time.



API DEMO

In the SDK package, users can find a Demo Application using the SDK APIs of GXV34X0, named **ApiDemo**.

