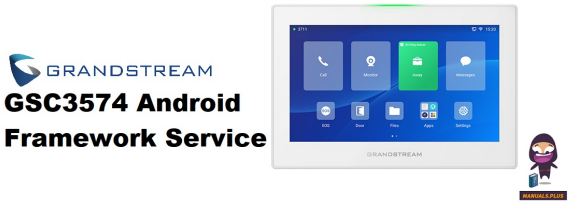
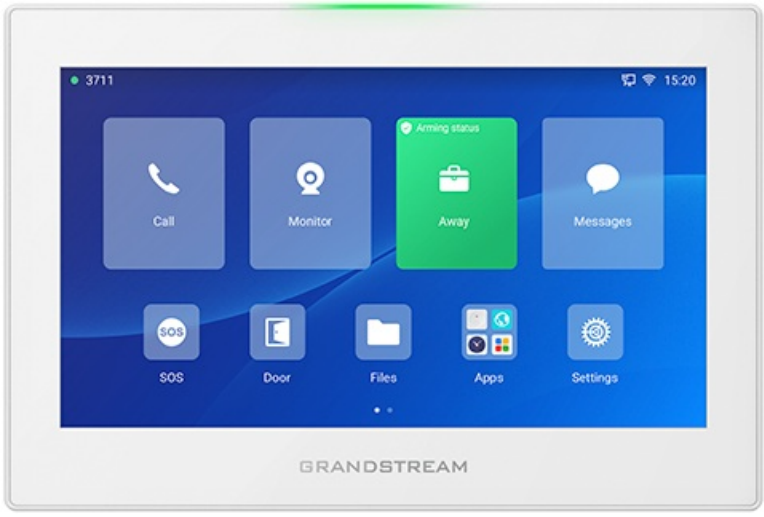


GRANDSTREAM GSC3574 Android Framework Service User Guide

Contents

GRANDSTREAM GSC3574 Android Framework Service



Product Usage Instructions

- Create a directory named gslibs under the App Module.
- Copy the JAR files gsapi-1.2.1.jar into the gslibs directory.
- In the build. gradle file, add dependencies using completely or provided:
- All APIs are named as XXXApi (e.g., BaseAccountApi, BaseCallApi).
- Before using the APIs, initialize ApiClient. For GSC3574/75, calling an API will automatically initialize it first. Use functions and monitors as follows:

FAQ

- **Q: Can I include the JAR files in the APK?**
  - **A:** No, please do not package the JAR files into the APK.
- **Q: Where can I find the latest firmware version for GXV34xx & GSC3574/3575?**
  - **A:** You can find the firmware release information at the following link: **Firmware Release Information**
- **Q: Are all classes, functions, and members of classes part of the supported API?**
  - **A:** No, only those defined in gsApiExternal-en are part of the supported API. Use others at your own risk as they may change or be deleted without notice.

OVERVIEW

GXV3470/GXV3480/GXV3450 and GSC3574/3575 operating systems is developed based onthe AndroidTM 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 GXV3470/GXV3480/GXV3450 and GSC3574/3575.

Note
Before starting the API demo or testing your own apps, please upgrade your GXV3470/GXV3480 /GXV3450/GSC3574/GSC3575 to the latest firmware version. The firmware release information can be found in the following link: **https://www.grandstream.com/support/firmware**

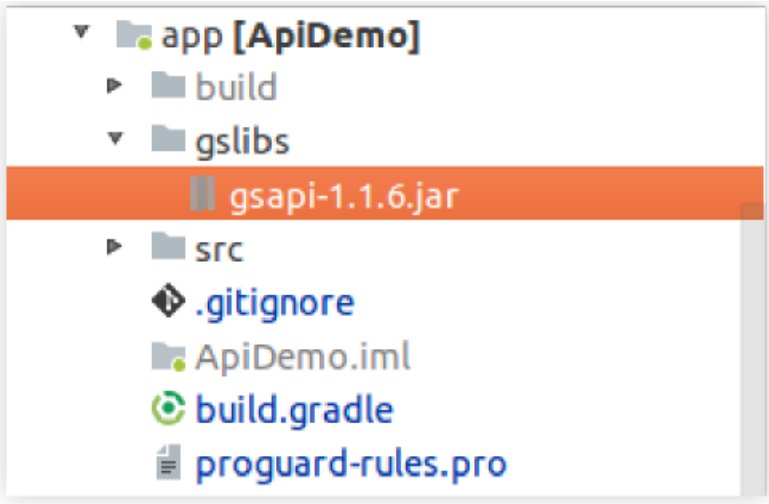
APPLICATION BUILDING

- With GS JAR files, users can start building the apps with Android Studio (or another IDE) following the steps in

this guide.

Create dir “gslibs” under App Module

- Under App module, create “slabs” directory. Then copy the JAR files “gsapi-1.2.1.jar” under the “gslibs” directory so the JAR files are reachable from the app files.



Create gslibs under App Module

Build.gradle
Add dependencies on the build.gradle file using compileOnly or provided. Please see the JAR files path below for reference:

- dependencies {. . . . . completely files (’gslibs/gsapi-1.2.1.jar’) }

Note
Please do not package the JARs into APK. Please refer to the below table that lists the JARs that can be included when compiling your app. They should be added in the “build.gradle” file if used

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.catalog	Support Gs Catalog 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:

API	Description
AudioRouteApi	APIs related to the audio route. This is used for switching the audio route and checking the audio route.
BaseAccountApi	APIs related to accounts. This is used for calls.
BaseLineApi	APIs related to lines. This is used for creating line channels for outgoing calls and incoming calls.
BaseCallApi	APIs related to calls. This is used for making a call, receiving a call, etc.
BasePhoneService	APIs for the target app as the default phone app.
ContactsContext	Useful API constants for Create-Read-Update-Delete (CRUD) contacts.
CalllogContext	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, etc.

Descriptions for APIs

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 risk. 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, and SmsMAnagerApi.
- Before using the APIs, users must initialize ApiClient first. For GSC3574/75, calling an Api (XXXApi) will automatically init it first so ApiClient/IApi are removed.
- Use a function such as XXXApi.function1()
- Use a monitor such as XXXApi.function2(callback2)
- Details of the API functions can be found under the gsApiExternal-en directory.

Initialize ApiClient
Supported Products: GXV3470, GXV3480, GXV3450.
The 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 the 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: GXV3470, GXV3480, GXV3450, GSC3574, GSC3575.
- 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 of 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.callback.destroy();
- mAccountStatusListener.callback = null; }

Monitor Account using AccountStatusListener

- private class MyAccountStatusListener extends AccountStatusListener {@Override public void onSipAccountStatusChanged(List<SipAccount> list, SipAccount sip account) {} }; AccountStatusListener mAccountStatusListener = null;

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

- public void update account() { BaseAccount account= BaseAccountApi.getAccountbyId(0);
- Log.d(TAG,”update account,original:”+account);
- if(account != null && account instanceof SipAccount){ account.setAccountName(”Test Account”);
- boolean ret = BaseAccountApi.updateSipAccount((SipAccount) account); if(ret){ Log.d(TAG,”update account,changed to:”+account); } } }

BaseCallApi

- Supported Products: GXV3470, GXV3480, GXV3450, GSC3574, GSC3575.
- Call functions allow users to make calls, end calls and monitor call status.

Make a Call
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> dialing Infos = 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

- public void endCall(View view) { DemoApplication app = (DemoApplication) getApplication();
- int lineId = app.getCurLineId();
- BaseCallApi.endCall(lineId); }

Monitor Incoming Call

- In order to answer an incoming call, users need to 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 that 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.callback.destroy();
- mCallStatusListener.callback = null; }

Monitor Call Status using CallStatusListener

- private class MyCallStatusListener extends CallStatusListener { @Override
- public void online status changed(int notifyType, BaseLine baseline, List<Baseline> list) { } @Override
- public void onLineIdChanged(int oldLineId, int newLineId) { } } private CallStatusListener mCallStatusListener = null;

Monitor Handset Hook ON/OFF

- Similar to monitoring the incoming calls, 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.callback.destroy();
- mHookStatusListener.callback = 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: GXV3470, GXV3480, GXV3450
- Users can perform blind transfer and attended transfer through transferBlind/transferAttended. The current call needs to be held before the transfer.
- After blind transfer, the current call ends directly, and the transferred party directly calls the transfer target.
- After the attended transfer, the current call will not end 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 canceled/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
Multiple phone applications are allowed in the 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 their own phone application and make it the default phone APP.

Implement Customized Phone Service
Customizing your own phone service requires extending BasePhoneService and handling on-hook/off-hook events.

Note
Gs JARs currently do not support subscriptions 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 the System Phone line for making calls anymore

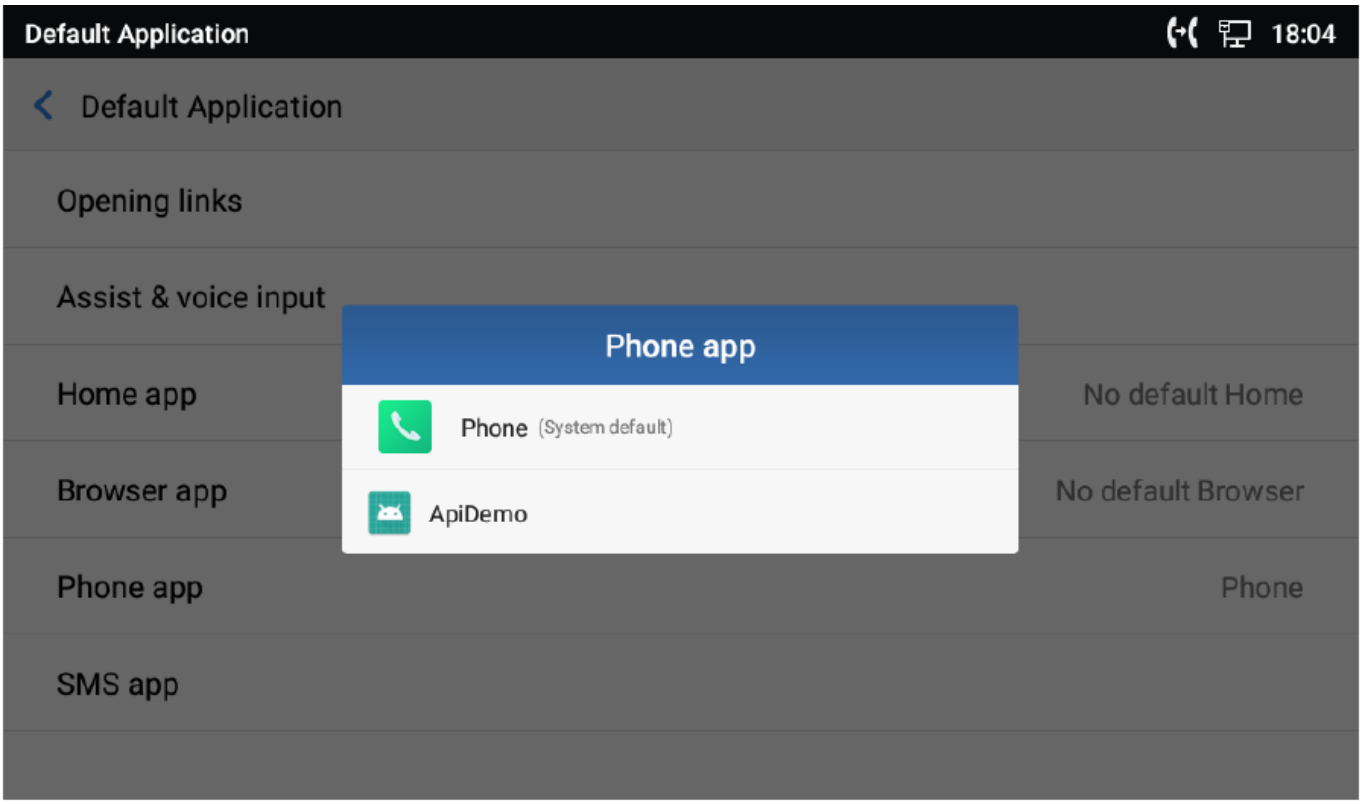
- public void onHookEvent(boolean OffHook) { super.onHookEvent(off-hook); //TODO: do something }
- public void onEhsHookEvent(boolean OffHook) { super.onEhsHookEvent(off-hook); //TODO: do something }
- public void onLineStateChanged(int lineId, int status) { super.onLineStateChanged(lineId, status); //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.onLineStateChanged(lineId, status); //TODO:do something return true; } }

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.



Default Phone app

SmsManagerApi

- Supported Products: GXV3470, GXV3480, GXV3450
- 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 numbers = “36324”;
- String content = “Test SMS to a single one.”;
- long msgId = SmsManagerApi.sendSmsAfterSaved(accountId, numberStr, content);
- if(msgId > -1){ Log.d(TAG,“Sending sms(“ + msgId + “)..”); } String group = number + “_” + “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(msg); 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.Receive SMS to get the content of the received SMS.

- vate SmsReceiver smsReceiver;
- private void registerSmsReceiver(){
- IntentFilter filter = new IntentFilter();
- filter.addiction(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.detection())){ 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 sms);
- Delete SMS based on sms;
- 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 remove type;
- 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: GXV3470, GXV3480, GXV3450, GSC3574, GSC3575.
With AudioRouteApi, the audio route can be switched or checked by the app.

Switch Audio Route
Use AudioRouteApi.switchVoiceToXXX to switch the 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 the hook.
- isEhsHeadsetConnected: check whether the EHS headset is connected or not.

Note

1. When the EHS headset is on the hook, EHS will not play any sound.
2. When calling setEhsHookStatus, the audio route must be on the 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 the speaker, even though the current voice channel is not on speaker.
To perform as above, the custom phone app should use the API switchCurrentVoiceWithSpeaker and switchCurrentVoiceWithoutSpeaker when the ringing is finished.
Below is a sample code when a new call is incoming.

- private class MyCallStatusListener extends CallStatusListener { @Override
- public void online status changed(int notifyType, BaseLine baseLine, List<BaseLine> list) { final BaseLine line = baseLine;
- if (line.getStatus() == 2) { // 2 means ringing if (CallSettingApi.isAlwaysRingSpeaker()) { AudioRouteApi.switchCurrentVoiceWithSpeaker();
- startline(); // start ringing }} else { if (more status == 2) { stopTone(); // stop ringing

- AudioRouteApi.switchCurrentVoiceWithoutSpeaker(); }} more status = line.getStatus(); }} CallSett

CallSettingApi

- Supported Products: GXV3470, GXV3480, GXV3450
- With CallSettingApi, the config status can be checked by the app.

DeviceApi
Supported Products: GXV3470, GXV3480, GXV3450, GSC3574, GSC3575.
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: GXV3470, GXV3480, GXV3450, GSC3574, GSC3575. 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 dial tone 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 playing dtmf tone
- public static void startDtmfTone(int keyValue);
- //Stop playing the 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 playing a busy tone
- public static void stopBusyTone();
- //Start play dial tone
- public static void startDialTone();
- //Stop play the dial tone
- public static void stopDialTone();
- //Start play second dial tone
- public static void startSecondDialTone();
- //Stop playing the 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 playing the ringtone in the DND mode
- public static void playDndTone();
- //Start playing the dial tone when there is an unread voicemail
- public static void startVMDialTone();
- //Stop playing the dial tone when there is an unread voicemail
- public static void stopVMDialTone();
- //Start play auto answer tone
- public static void startAutoAnswerTone();
- //Stop playing auto answer tone
- public static void stopAutoAnswerTone();
- //Gets the type of tone currently playing
- public static int getCurToneType();

Gs Core SDK

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

Device Manager

- Supported Products: GXV3470, GXV3480, GXV3450, GSC3574, GSC3575.

Get the device connected state.

- // is a 3.5mm earphone device connected
- GsDevStateManager.instance().isDeviceConnected(GsDevStateManager.NAME_EARPHONE);
- // is the device connected
- GsDevStateManager.instance().isDeviceConnected(GsDevStateManager.NAME_EHS);
- // is HDMI device connected
- GsDevStateManager.instance().isDeviceConnected(GsDevStateManager.NAME_HDMI);
- //Is HDMI in the 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
Supported Products: GXV3470, GXV3480, GXV3450, GSC3574, GSC3575.

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

Get Device MAC

- GsDevStateManager.instance().getDeviceMac();

CONTACTS

- Supported Products: GXV3470, GXV3480, GXV3450, GSC3574, GSC3575. android.content.ContentResolver has already provided Create-Read-Update-Delete (CRUD) APIs for operating contacts.
- Please refer to the below links for the API 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 the 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 the contacts database table ContactsContract.Data:

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

Contacts Contract.Data
Changes for ContactsContract.RawContacts

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

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

ContactsContract.RawContacts
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 as “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 add contact(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, accounted)

- .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 { content resolver.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 update contact(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 { content resolver.applyBatch(ContactsContract.AUTHORITY, ops);
- } catch (Exception e) { e.printStackTrace(); } }

Read Contacts
Here is an example of reading contacts and manipulating all the parameters:

- public void get contacts() { 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 = content resolver
- .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 contacted) { 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
- + "=" + contacted, null) .build());
- try { content resolver.applyBatch(ContactsContract.AUTHORITY, ops);
- } catch (Exception e) { e.printStackTrace(); } }

CALL LOGS

- Supported Products: GXV3470, GXV3480, GXV3450, GSC3574, GSC3575.
- android.content.ContentResolver has already provided Create-Read-Update-Delete(CRUD) APIs for operating catalogs.
- Please refer to the below link for the API information:
- **https://developer.android.google.cn/reference/android/content/ContentProvider**
- For more details about catalog 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 functions. Using android .content.ContentResolver to CRUD catalogs should be able to meet most of the needs.

IMPORTANT
Please make sure to read the below content for the changes applied to 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.

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

Create a Call Log
As the call log is closely related to the 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, accounted)
- .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_CONFERENCE,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 = content resolver.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 of updating 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 {
- content resolver.applyBatch(CallLog.AUTHORITY, ops);
- } catch (Exception e) { e.printStackTrace(); } }

Delete call logs
Here is an example of deleting 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 { content resolver.applyBatch(CallLog.AUTHORITY, ops);
- } catch (Exception e) {
- e.printStackTrace(); } }

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

- int color = 0xFF00FF00;
- int on Ms = 1000;
- int off Ms = 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 GXV3470, GXV3480, GXV3450, GSC3574, and GSC3575. 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.

Product	**GXV3470**	**GXV3480**	**GXV3450**	**GSC3574**	**GSC3575**
Red LED	On/Off	On/Off	On/Off	On/Off	On/Off
Green LED	On/Off	On/Off	On/Off	On/Off	On/Off
Blue LED	N/A	N/A	N/A	N/A	N/A
Keep on frequency	on Ms: 1000, off Ms: 0	on Ms: 1000, off Ms: 0	on Ms: 1000, off Ms: 0	on Ms: 1000, off Ms: 0	on Ms: 1000, off Ms: 0
Slow splash frequency	on Ms: 1000, off Ms: 3000	on Ms: 1000, off Ms: 3000	on Ms: 1000, off Ms: 3000	on Ms: 1000, off Ms: 3000	on Ms: 1000, off Ms: 3000
Fast splash frequency	on Ms: 1000, off Ms: 1000	on Ms: 1000, off Ms: 1000	on Ms: 1000, off Ms: 1000	on Ms: 1000, off Ms: 1000	on Ms: 1000, off Ms: 1000

LED Colors and Frequencies

GsLightsManager

- Supported Products: GXV3470, GXV3480, GXV3450.

- Except for controlling LED with Notification, the GsLightsManager provided functions to control the LED freely. The following are method definitions:
- public void open custom(int color, int shineType);
- public void close custom();
- 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.

LedApi
Supported Products: GSC3574, GSC3575.
LedApi allows to control of LED on GSC3574/75 using the following methods:

- public void openTopCustom(LedApi.Color color, LedApi.ShineType shineType);
- public void closeTopCustom();
- public void openSideCustom(LedApi.Color color, LedApi.ShineType shineType);
- public void closeSideCustom();

The parameters:

- shineType: controls the shining type of the custom LED that the app wants to set (such as High Speed Flashing).
- color: controls the color of the custom LED that the app wants to set (Red or Green).

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 the extension module click event by using MpkApi.addStatusListener and MpkApi.removeStatusListener.
- Here is an example on how to use the monitor extension module to click event APIs.

Start Ext Click Event Monitor

- private void initExtClickListener() { mMpkStatusListener = new MpkStatusListener();
- 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;

Control settings button in taskbar via API

- To control whether to display the settings button on the device taskbar, users could use the below API commands to hide/display it and get the display status.

Send a broadcast to hide the settings button

- Intent intent = new Intent(“com.android.systemui.settings_button.visibility”);
- intent.putExtra(“visibility”, false);
- send broadcast(intent);

Send a broadcast to the display settings button

- Intent intent = new Intent(“com.android. systemui.settings_button.visibility”);
- intent.putExtra(“visibility”, true);
- send broadcast(intent);

Get hide status (1 – display; 0 – hide)

- Settings.Global.getInt(getContentResolver(), “settings_button_visibility”, 0))

RoomPanelApi
Supported Products: GSC3574, GSC3575.
RoomPanelApi allows to obtain and modify information about Conference Rooms using the following methods:

- public static boolean startConfReservation(int time);
- public static boolean extendConfReservation(int time);
- public static boolean endConfReservation();
- public static ConfReservation getConfReservationInfo();

The “time” parameter (in minutes) can refer to the conference start time or extended time.

DEVELOP APPS WITH ADB

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

Enable Developer Mode on Device

- Please enable developer mode under the 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 the below content is displayed, it means the connection is successful:

- List of devices attached
- 192.168.100.1:5555 de

ADB Disconnect

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

Note
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 and GSC3574/75, named ApiDemo.

CHANGE LOG
This section documents significant changes from previous versions of the SDK Framework Service guide for GXV3470/GXV3480/GXV3450/GSC3574/GSC3575. Only major new features or major document updates are listed here. Minor updates for corrections or editing are not documented here.

Firmware Version 1.3.1
Product Name: GSC3574 / GSC3575

- Added support for GSC3574/75 SDK (1.0.1.1) inherited from GXV34xx SDK.
- Updated SDK init logic: calling XXXApi will automatically init it for first use. [USE APIs]
- Added special API for GSC357x: RoomPanelApi. [RoomPanelApi]
- Replaced GsLightManager with LedApi. [LedApi]
- Added support for Android. content.ContentResolver to query GS custom contacts and call logs. [CONTACTS] [CALL LOGS]

- Added support for BaseAccountApi, BaseCallApi, BaseLineApi, DeviceApi, CallToneApi, AudioRouteApi,
- GsDevStateManager, DndManager. [BaseAccountApi] [BaseCallApi] [BaseLineApi] [DeviceApi] [CallToneApi] [AudioRouteApi] [GsDevStateManager] [DndManager]

Firmware Version 1.0.1.29
Product Name: GXV3480 / GXV3450 / GXV3470

- Added support to remove the Settings button in the Taskbar via API. [CONTROL SETTINGS BUTTON IN TASKBAR VIA API]

SUPPORTED DEVICES

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

Model	Supported	Firmware
GXV3470	Yes	1.0.1.15 or Higher
GXV3480	Yes	1.0.1.15 or Higher
GXV3450	Yes	1.0.1.15 or Higher
GSC3574	Yes	1.0.1.7 or higher
GSC3575	Yes	1.0.1.7 or higher

Documents / Resources

	GRANDSTREAM GSC3574 Android Framework Service [pdf] User Guide GXV3470, GXV3480, GXV3450, GSC3575, GSC3574 Android Framework Service, GSC3574, Android Framework Service, Framework Service, Service

References

- ContentProvider | Android Developers
- CallLog | Android Developers
- ContactsContract | Android Developers
- User Manual