Grandstream Networks, Inc.

GXV3275 SDK Framework Service Guide v3.6

User Guide

COPYRIGHT

The latest electronic version of this guide is available for download here:
http://www.grandstream.com/support

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 adaptor with devices as it may cause damage to the products and void the manufacturer warranty.

Table of Content

Table of Tables

OVERVIEW

Introduction

GXV3275 operating system is developed based on the Android™ platform. Besides inheriting the Android interface functions, it has added other interfaces according to users' requirements. This document describes how to use GXV3275 APIs for users' application development.

In this package, users will find the following useful information in the three folders:

Table 1: GXV3275 SDK Framework Service Package content

Folder	Content	Description
Doc	GXV3275 SDK Framework Service Guide	Document describing how to use GXV3275 APIs for user's application
Sample	ApiDemo.apk	Demo app to install on GXV3275
	Android.jar	To replace the file in Android SDK package for GXV3275
code	Source code of GXV3275 ApiDemo.apk	

SDK Version Number

For each API, the change log information is specified in this document. Users could get the SDK version number via the following methods.

- Import class
  import com.base.module.sdk.Version;

- Example: get version number
  int version = Version.SDK_VERSION; **//int type, default value is 1**

Note:

Before starting the API demo or testing your own apps, please upgrade your GXV3275 to the latest firmware version. The firmware release information can be found in the following link:
http://www.grandstream.com/support/firmware

CALL API

Call API Actions

Added in SDK version 1

Call API is inherited from Android™ operating system standard Call API, mainly controlled by the following two types of Action.

- **android.intent.action.DIAL**
  Description: Open call screen to edit number before dialing out.

- **android.intent.action.CALL**
  Description: Dial out.

Call API Parameters

Added in SDK version 1

When using Call API, users could specify parameters for the action. The parameters are stored as "key-value" where key is a string and value can be used as different types. For example:

- **key-value:** key-values are all string type.
- **value:** replaced by the account ID (from 0 to 5 for account 1 to 6), int type.
- **account:** int value.

Open Dial Panel

Added in SDK version 1

Table 2: Open Dial Panel

Public Method	**Intent intent = newIntent(Intent.ACTION_DIAL); intent.setData(Uri.parse("tel:")); startActivity(intent);**
Description	To open the dial panel
Parameters	N/A
am command	am start -a android.intent.action.DIAL -d tel:
Return	Enter dial screen for users to edit number

Added in SDK version 1

Table 3: Edit Number Before Dialing

Public Method	**Intent intent = new Intent(Intent.ACTION_DIAL); intent.setData(Uri.parse("tel:"+String phoneNumber)); intent.putExtra("account", int accountID); startActivity(intent);**

Description	To edit number before dialing out
Parameters	• phoneNumber: the number to be dialed • accountID: the account ID (from 0 to 5 for account 1 to 6) on the phone
am command	am start -a android.intent.action.DIAL -d tel: phoneNumber --ei account accountID
Return	Enter dial screen for users to edit number

Dial Out

Added in SDK version 1

Table 4: Dial Out

Public Method	**Intent intent = new Intent(Intent.ACTION_CALL);
	intent.setData(Uri.parse("tel:"+String phoneNumber));
	intent.putExtra("account", int accountID); startActivity(intent);
	Intent.putExtra("isVideo", boolean isVideo); (可选)**

Description	To dial out
Parameters	• phoneNumber : the number to be dialed • accountID: the account ID (from 0 to 5 for account 1 to 6) on the phone • "isVideo": **true**/**false**. Default value is **false**, which means audio call. If set to **true**, the call will be dialed out as video call
am command	am start -a android.intent.action.CALL -d tel: phoneNumber --ei account accountID
Return	Enter dial screen for users to edit number

Added in SDK version 1

Table 5: Redial

Public Method	**Intent intent = new Intent(Intent.ACTION_CALL); intent.setData(Uri.parse("tel:redial")); startActivity(intent);**
Description	To redial the last dialed number
Parameters	N/A
am command	am start -a android.intent.action.CALL -d tel:redial
Return	Dialing out the last dialed number

Managing Call Status

This function is added in SDK version 2.

Users could check the current call status on the GXV3275 by **CallStatusManager** class. The following status can be obtained:

- The GXV3275 is in call screen or not.
- The line is busy or not.
- Calling/talking status of a specific account on the GXV3275.
- It can also be used to bind or unbind to **PhoneStatusService** function.

Please follow the instructions below to use the **CallStatusManager** class in GXV3275 SDK API.

- Import **CallStatusManager** class.

  Use the following code to import **CallStatusManager** class first.
  import com.base.module.phone.service.CallStatusManager

- Create an instance.

  Use method **CallStatusManager.instance ()** to create an instance.

- Bind to **PhoneStatusService**.

  Use the instance created in the above step 2 to call **bindPhoneService (Context context)** method in **CallStatusManager** class to bind to the service. Then users can call the other methods in **CallStatusManager** class.

- Check the call status using the method of **CallStatusManager** class, listed in table 5 and table 6.

- Unbind to **PhoneStatusService**.

  Once the application process is done, users could call **unbindPhoneService (Context context)** method in **CallStatusManager** class to unbind to the service.

CallStatusManager class has the following methods:

Added in SDK version 2

Table 6: CallStatusManager Class Methods

Public Method	CallStatusManager.instance()
Function	Create a CallStatusManager instance
Parameters	N/A
am command	N/A
Return	A **CallStatusManager** instance will be returned

Public Method	boolean isCallViewShow()
Function	Check if the call screen is currently displayed or not
Parameters	N/A
am command	N/A
Return	**true**/**false** for currently displayed/not displayed
Public Method	**boolean isBusy()**
Function	Check if the line is busy at the moment
Parameters	N/A
am command	N/A
Return	**true**/**false** for busy/not busy
Public Method	**void bindPhoneService(Context context)**
Function	Bind **context** to **PhoneStatusService**
Parameters	context
am command	N/A
Return	N/A
Public Method	**void unbindPhoneService(Context context)**
Function	Unbind **context** to **PhoneStatusService**
Parameters	context
am command	N/A
Return	N/A
Public Method	**int getLineStatus(int line)**
Function	Obtain the calling/talking status of the specified line
Parameters	0 to 5 for line 1 to line 6
am command	N/A
Return	The following result can be returned, depending on the actual line status: **public static final int STATUS_IDLE = 0;** **public static final int STATUS_DIALING = 1;** **public static final int STATUS_RINGING = 2;** **public static final int STATUS_CALLING = 3;** **public static final int STATUS_CONNECTED = 4;** **public static final int STATUS_ONHOLD = 5;** **public static final int STATUS_TRANSFERED = 6;** **public static final int STATUS_ENDING = 7;** **public static final int STATUS_FAILED = 8;** **public static final int STATUS_TRANSFER = 9;** **public static final int STATUS_CONFERENCE = 10;** **public static final int STATUS_PAGING = 11;** **public static final int STATUS_RINGBACK = 12;** **public static final int STATUS_IPCALL = 13;**

Added in SDK version 3

Table 7: CallStatusManager Class Methods 2

Public Method	LineObj getLineObj(int line)
Function	Get line object, including all information of line
Parameters	line
am command	N/A
Return	Specific line object
Public Method	**LineObj[ ] getAllLineObjs()**
Function	Get the array sets of all lines status
Parameters	N/A
am command	N/A
Return	The array sets of all lines objects
Public Method	**void setOnConnectServiceListener(OnConnectServiceListener listener)**
Function	Set listening the status of service of CallStatusManager, OnConnectServiceListener will be introduced in this document later
Parameters	N/A
am command	N/A
Return	N/A
Public Method	**void setOnPhoneStatusListener(IPhoneStatusListener.Stub() listener)**
Function	Set call status listener, IPhoneStatusListener will be introduced in this document later. **Note: When receiving OnConnectServiceListener, it will take effect after a successful connection callback setting.**
Parameters	listener
am command	N/A
Return	N/A
Public Method	**void removePhoneStatusListener(IPhoneStatusListener listener)**
Function	Remove call status listener

Parameters	listener
am command	N/A
Return	N/A
Public Method	**void endCall(int line)**
Function	Stop the call of specific line
Parameters	line
am command	N/A
Return	N/A
Public Method	**void endAllCall()**
Function	Stop all calls
Parameters	N/A
am command	N/A
Return	N/A
Public Method	**boolean startRecord()**
Function	Start to record calls
Parameters	N/A
am command	N/A
Return	true: starting successfully false: starting failed, e.g. there is no “on calling” status
Public Method	**void stopRecord()**
Function	Stop calls recording
Parameters	N/A
am command	N/A
Return	N/A

OnConnectServiceListener interface below：

```
public interface OnConnectServiceListener{
        void onConnected(boolean isConnect);
        //isConnect true is the connection service is successful, false is the connection service is failed.
}
```

IPhoneStatusListener interface below：

```
public interface IPhoneStatusListener {
    void onPhoneStatusChanged(int line, int state, int accountID);
    //line line id  state line status   accountID Account ID
    void onRecordStatueChanged(boolean isRecording);
    //isRecording true Recording false Stop recording
}
```

LineObj Class Interfaces

Import LineObj Class：

- **import com.base.module.phone.service.LineObj;**

This class is to describe the details of a line, you can get all information on a line through the class, and the main methods are described on following table.

Added in SDK version 3

Table 8: Lines Status Interfaces

Public Method	int getState()
Function	Get lines status
Parameters	N/A
am command	N/A
Return	The following result can be returned, depending on the actual line status: **public static final int STATUS_IDLE = 0;** **public static final int STATUS_DIALING = 1;** **public static final int STATUS_RINGING = 2;** **public static final int STATUS_CALLING = 3;** **public static final int STATUS_CONNECTED = 4;** **public static final int STATUS_ONHOLD = 5;** **public static final int STATUS_TRANSFERED = 6;** **public static final int STATUS_ENDING = 7;** **public static final int STATUS_FAILED = 8;** **public static final int STATUS_TRANSFER = 9;** **public static final int STATUS_CONFERENCE = 10;** **public static final int STATUS_PAGING = 11;** **public static final int STATUS_RINGBACK = 12;** **public static final int STATUS_IPCALL = 13;**
Public Method	**int getAccountNumber()**
Function	Get account ID
Parameters	N/A
am command	N/A
Return	Account ID (0-5)

Public Method	String getCallerNumber()
Function	Get incoming call number
Parameters	N/A
am command	N/A
Return	Incoming call number, string type
Public Method	**String getCallerName()**
Function	Get incoming call name
Parameters	N/A
am command	N/A
Return	Incoming call name, string type
Public Method	**Bitmap getCallerIcon()**
Function	Get incoming call icon
Parameters	N/A
am command	N/A
Return	Incoming call icon, bitmap type
Public Method	**int getLineId()**
Function	Get line ID
Parameters	N/A
am command	N/A
Return	Line ID （0-7）
Public Method	**String getDtmfStr()**
Function	The dtmf which has already been sent by current line
Parameters	N/A
am command	N/A
Return	DTMF, string type, e.g. “123”
Public Method	**boolean isInConference()he**
Function	Determine whether the line is Conference line
Parameters	N/A
am command	N/A
Return	true (Conference line) / false (Normal line)
Public Method	**boolean isOnMute()**
Function	Determine whether the line is on mute（no local mic）
Parameters	N/A

am command	N/A
Return	true (on mute)/ false (not on mute)
Public Method	**boolean isRemoteOnMute()**
Function	Determine whether the remote party is on mute （Conference room， the voice line is on mute in Conference room）
Parameters	N/A
am command	N/A
Return	true (not allowed)/ false (allowed)
Public Method	**boolean isSrtp()**
Function	Determine whether enable srtp stream
Parameters	N/A
am command	N/A
Return	true (enable)/ false (disable)
Public Method	**boolean isVideo()**
Function	Determine whether enable video
Parameters	N/A
am command	N/A
Return	true (video)/ false (audio)
Public Method	**boolean isVideoComming()**
Function	Determine whether enable video incoming calls
Parameters	N/A
am command	N/A
Return	**true** (video incoming calls)/ **false** (audio incoming calls)

Handset Status and Handset Detection

Added in SDK version 2

Users could disable the handset as well as detect handset status by sending specific broadcast message.

- The following code can be used to disable the handset:

```
Intent intent = new Intent("com.base.module.phone.SKIPHOOK");
intent.putExtra("skiphook", true);   //"true": disable the handset; "false": enable the handset
sendBroadcast(intent);
```

- The following code can be used to detect handset status:

To obtain the handset status, users need to monitor the broadcast with action **"com.base.module.phone.HOOKEVENT"** and then get the key value for "**hookoff**" from the intent of the broadcast.

```
boolean hookoff = intent.getBooleanExtra("hookoff", false);
```

If the key value of "**hookoff**" is "true", the handset is offhook. If the key value of "**hookoff**" is "false", the handset is onhook.

THIRD-PARTY PHONE API

Phone_Common_v1.jar

Phone_common_v1.jar is compatible with firmware version 1.0.3.37 and older versions. And it's supported with firmware version 1.0.3.64 and newer versions.

Added in SDK version 5

- **Initialization**

Import the classes as follows:

```
import com.base.module.phone.base.LineStatusBase;
import com.base.module.phone.event.EventManager;
import com.base.module.phone.manager.AudioRoutePath;
import com.base.module.phone.manager.BroardcastManager;
import com.base.module.phone.manager.LineStatusBinder;
import com.base.module.phone.manager.PhoneAudioManager;
```

Initialize the instance as follows:

```
public void onCreate(){
    AudioRoutePath.instance().setContext(this);
    LineStatusBase.instance();
    LineStatusBinder.instance().setContext(this);
    PhoneAudioManager.instance().setContext(this);
    BroardcastManager.instance().init(this);
    EventManager.setContext(this);
    PhoneAudioManager.instance().registerModeController(this.getPackageName());
}
```

Table 9: Third Party Phone Initialization V1

Public Method	**AudioRoutePath.instance().setContext(this);**
Description	Create an audio route path instance
Parameters	N/A
am command	N/A
Return	N/A
Public Method	**LineStatusBase.instance();**
Description	Create a line status base instance
Parameters	N/A
am command	N/A
Return	N/A
Public Method	**LineStatusBinder.instance().setContext(this);**
Description	Create a line status binder instance

Parameters	N/A
am command	N/A
Return	N/A
Public Method	**PhoneAudioManager.instance().setContext(this);**
Desciption	Create a phone audio manager instance
Parameters	N/A
am command	N/A
Return	N/A
Public Method	**BroardcastManager.instance().init(this);**
Description	Create a broadcast manager instance
Parameters	N/A
am command	N/A
Return	N/A
Public Method	**EventManager.setContext(this);**
Description	Create an event manager instance
Parameters	N/A
am command	N/A
Return	N/A
Public Method	**PhoneAudioManager.instance().registerModeController(this.getPackageName());**
Description	Create a third-party phone register instance
Parameters	N/A
am command	N/A
Return	N/A

- **Call Status Callback**

Users call the methods in GuiEventUIHelper class to listen to the call status.

Call steps:

1. Import the class:
   import com.base.module.phone.base.GuiEventUIHelper;
   import com.base.module.phone.base.GuiEventUIHelper.OnPhoneEventListener;
2. Create an instance:
   Call the method GuiEventUIHelper.instance() to create an instance.
3. Add the interface:

Call the method GuiEventUIHelper.instance().setOnPhoneEventListener(new OnPhoneEventListener());

Implement the instance:

```
GuiEventUIHelper.instance().setOnPhoneEventListener(
        new OnPhoneEventListener() {

                @Override
                public void onHandlePhoneEvent(String eventName,
                                        Object... args) {
                        startMainActivity();
                        if(null != mDbusEnventListener) {
                                mDbusEnventListener.onHandlePhoneEvent(eventName, args);
                        }
                }
        });
```

Table 10: Call Status Listener Interfaces V1

Public Method	**GuiEventUIHelper.instance()**
Description	Create a GuiEventUIHelper instance
Parameters	N/A
am command	N/A
Return	Return a GuiEventUIHelper instance
Public Method	**onHandlePhoneEvent(String eventName,Object... args)**
Description	Inform the current call status on application layer
Parameters	eventName=call: the change of the call status args: int32 state int32 line int32 account int32 msg string phoneNumber string callerName state: the status in a call state=0: STATUS_IDLE state=1: STATUS_DIALING state=2: STATUS_RINGING state=3: STATUS_CALLING state=4: STATUS_CONNECTED state=5: STATUS_ONHOLD state=7: STATUS_ENDING state=8: STATUS_FAILED line: the line used by the current call assigned by lower layer. Every line has a call ID. Each call is based on the line. Account: the account in use msg: specific parameter that differs under different status: msg=0,state=2: audio call ringing msg=1,state=2: video call ringing (msg&0x10) != 0, state=4: video call connected (msg&0x20) != 0, state=4: video call connected (msg&0x30) != 0, state=4: video call connected msg=4,state=8: NORESPONSE

	msg=5,state=8: 403 Forbbiden msg=6,state=8: 404 NOT FOUND! msg=7,state=8: Request Timeout msg=8,state=8: 486 Busy! msg=9,state=8: 488 Not Acceptable Here! msg=10,state=8: Number does not match dial plan msg=15,state=8: 503 Service Unavailable msg=16,state=8: 500 Internal Server Error msg=17,state=8: 501 Not Implemented msg=18,state=8: 502 Bad Gateway msg=19,state=8: 504 Gateway Timeout msg=20,state=8: 513 Message too Large msg=21,state=8: 600 Busy Everywhere msg=22,state=8: 603 Decline msg=23,state=8: 604 Does Not Exist Anywhere msg=24,state=8: 606 Not Accaptable phoneNumber: current number that in the call in/out callerName: the account name of the phoneNumber
am command	N/A
Return	N/A

- **Phone Keys**

By calling the methods in GuiEventUIHelper class, users can only listen to the keys on the phone. Other keys' listeners still follow Android official API.

Call steps:

1. Import the class:
   import com.base.module.phone.base.GuiEventUIHelper;
   import com.base.module.phone.base.GuiEventUIHelper.OnPhoneEventListener;

2. Create an instance:
   Call the method GuiEventUIHelper.instance() to create an instance.

3. Add the interface:
   Call the method GuiEventUIHelper.instance().setOnPhoneEventListener(new OnPhoneEventListener());

Implement the instance:

```
GuiEventUIHelper.instance().setOnPhoneEventListener(
        new OnPhoneEventListener() {

                @Override
                public void onHandleKeyEvent(int keyCode) {

                }
});
```

Table 11: Phone Key Listener Interface V1

Public Method	onHandleKeyEvent(int keyCode)
Description	Listen to the phone keys
Parameters	keyCode: keyCode values and features

	keyCode =91 Mute keyCode =1800: Contact keyCode =1802: Headset keyCode =1803: Voicemail keyCode =1804: Transfer keyCode =1806: Dial keyCode =1807: Hands free keyCode =10001: Handset off-hook keyCode =10002: Handset on-hook
am command	N/A
Return	N/A

Phone_Common_v2.jar

Phone_common_v2.jar is compatible with firmware version 1.0.3.42 and newer versions.

- **Initialization**

Import the classes as follows:

```
import com.base.module.phone.demo.AudioRoutePath;
import com.base.module.phone.demo.BroardcastManager;
import com.base.module.phone.demo.EventManager;
import com.base.module.phone.demo.LineStatusBase;
import com.base.module.phone.demo.LineStatusBinder;
import com.base.module.phone.demo.PhoneAudioManager;
```

Initialize the instance:

```
public void onCreate(){
      AudioRoutePath.instance().setContext(this);
      LineStatusBase.instance();
      LineStatusBinder.instance().setContext(this);
      PhoneAudioManager.instance().setContext(this);
      BroardcastManager.instance().init(this);
      EventManager.setContext(this);
      PhoneAudioManager.instance(). registerModeController(this.getPackageName());
}
```

Table 12: Third Party Phone Initialization V1

Public Method	**AudioRoutePath.instance().setContext(this);**
Description	Create an audio route path instance
Parameters	N/A
am command	N/A
Return	N/A
Public Method	**LineStatusBase.instance();**
Description	Create a line status base instance
Parameters	N/A

am command	N/A
Return	N/A
Public Method	**LineStatusBinder.instance().setContext(this);**
Description	Create a line status binder instance
Parameters	N/A
am command	N/A
Return	N/A
Public Method	**PhoneAudioManager.instance().setContext(this);**
Description	Create a phone audio manager instance
Parameters	N/A
am command	N/A
Return	N/A
Public Method	**BroardcastManager.instance().init(this);**
Description	Create a broadcast manager instance
Parameters	N/A
am command	N/A
Return	N/A
Public Method	**EventManager.setContext(this);**
Description	Create an event manager instance
Parameters	N/A
am command	N/A
Return	N/A
Public Method	**PhoneAudioManager.instance().registerModeController(this.getPackageName());**
Description	Create a third-party phone register instance
Parameter	N/A
am command	N/A
Return	N/A

- **CALL STATUS CALLBACK**

Users call the methods in GuiEventUIHelper class to listen to the call status.

Call steps:

1. Import the class:

```
import com.base.module.phone.demo.GuiEventUIHelper;
import  com.base.module.phone.demo.GuiEventUIHelper.OnPhoneEventListener;
```

2. Create an instance:

   Use the method GuiEventUIHelper.instance() to create an instance.

3. Add the interface:

   Use the method GuiEventUIHelper.instance().setOnPhoneEventListener(new OnPhoneEventListener());

Implement the instance:

```
GuiEventUIHelper.instance().setOnPhoneEventListener(
                                new OnPhoneEventListener() {

    @Override
    public void onRinging(String callerNumber, String callerName) {
        // TODO
    }

    @Override
    public void onCalling(String callNumber) {
        // TODO
    }

    @Override
    public void onConnected() {
        // TODO
    }

    @Override
    public void onDialing() {
        // TODO
    }

    @Override
    public void onEnding() {
        // TODO
    }

    @Override
    public void onFailed() {
        // TODO
    }

    @Override
    public void onHold() {
        // TODO
    }

    @Override
    public void onIdle() {
        // TODO
    }
});
```

Table 13: Call Status Listener Interfaces V2

Public Method	GuiEventUIHelper.instance()
Description	Create a GuiEventUIHelper instance
Parameters	N/A
am command	N/A
Return	A GuiEventUIHelper instance
Public Method	**void onIdle()**
Description	Idle or complete call
Parameters	N/A
am command	N/A
Return	N/A
Public Method	**void onDialing()**
Description	On dialing. May not be used
Parameters	N/A
am command	N/A
Return	N/A
Public Method	**void onRinging(String callerNumber, String callerName)**
Description	A call is ringing
Parameters	callerNumber: the caller number callerName: the caller name
am command	N/A
Return	N/A
Public Method	**void onCalling(String callerNumber)**
Description	On calling
Parameters	callerNumber: the caller number
am command	N/A
Return	N/A
Public Method	**void onConnected()**
Description	The call is connected
Parameters	N/A
am command	N/A
Return	N/A

Public Method	void onHold()
Description	Status can be on hold or off hold
Parameters	N/A
am command	N/A
Return	N/A
Public Method	**void onEnding()**
Description	The call is ending. This may not be used
Parameters	N/A
am command	N/A
Return	N/A
Public Method	**void onFailed()**
Description	Call failed
Parameters	N/A
am command	N/A
Return	N/A

- **Phone Keys**

By calling the methods in GuiEventUIHelper class, users can only listen to the keys on the phone. Other keys' listeners still follow Android official API.

- **Call steps:**

1. Import the class:
   import com.base.module.phone.base.GuiEventUIHelper;
   import com.base.module.phone.base.GuiEventUIHelper.OnPhoneEventListener;

2. Create an instance:
   Call the method GuiEventUIHelper.instance() to create an instance.

3. Add the interface:
   Call the method GuiEventUIHelper.instance().setOnPhoneEventListener(new OnPhoneEventListener());

```
GuiEventUIHelper.instance().setOnPhoneEventListener(
        new OnPhoneEventListener() {

                @Override
                public void onHandleKeyEvent(int keyCode) {

                }
});
```

Table 14: Phone Key Listener Interfaces V2

Public Method	onHandleKeyEvent(int keyCode)
Description	Listen to the phone keys
Parameters	keyCode: keyCode values and features keyCode =91 Mute keyCode =1800: Contact keyCode =1802: Headset keyCode =1803: Voicemail keyCode =1804: Transfer keyCode =1806: Dial keyCode =1807: Hands free keyCode =10001: Handset off-hook keyCode =10002: Handset on-hook
am command	N/A
Return	N/A

MESSAGE API

Message API is mainly controlled by the following action:

- **android.intent.action.SENDTO**

This action on GXV3275 has three new fields defined in addition to the message sending action on the Android platform. The three new fields are as follows:

- Account on GXV3275
- Enter message editing window or not
- Insert message to draft box or not

Those fields are used in different functions in Message API according to the parameters.

Message API Parameters

Added in SDK version 1

The three parameters are stored as "key-value" where key is a string, and value can be used as different types:

- **key-value**: key, String type;
  value: replaced by **true** or **false**. The default value is **false**, Boolean type.
  This parameter controls whether the phone will open Message editing window.

- "**account"-int value:**
  Description: **"account"**: key, String type;
  value: replaced by account ID (from 0 to 5 for account 1 to 6) on the phone, int type.

- **boolean value**:
  Description: Determine whether the message will be inserted to Draft box, the default is false.

⚠ **Note:**

"**draft**" has higher priority to "**editable**". When the 3rd party sets both "**editable**" and "**draft**" as **true**, the Message will be inserted to Draft box (instead of showing editing window).

Open Message Editing Window

Added in SDK version 1

Table 15: Open Message Editing Window

Public Method	**Uri uri = Uri.parse("smsto:" + phoneNumber);** **Intent intent = new Intent(Intent.ACTION_SENDTO,uri);** **intent.putExtra("sms_body",content);** **intent.putExtra("editable",true); intent.putExtra("draft",false);** **intent.putExtra("account",int accountID); startActivity(intent);**
Description	To open the Message editing window
Parameters	key-value: "editable"-**true** key-value: "draft"-**false** String content: the message content to be sent (optional) String phoneNumber: the number to send message to (optional) int accountID: Account ID(0-5)
am command	am start -a android.intent.action.SENDTO -d smsto:phoneNumber --es sms_body content --ei account accountID --ez editable true --ez draft false
Return	Enter the Message editing window

Send Message

Added in SDK version 1

Table 16: Send Message

Public Method	**Uri uri = Uri.parse("smsto:" + phoneNumber);** **Intent intent = new Intent(Intent.ACTION_SENDTO,uri);** **intent.putExtra("sms_body",content); intent.putExtra("editable",false);** **intent.putExtra("draft",false);** **intent.putExtra("account",int accountID); startActivity(intent);**
Description	To send message
Parameters	key-value: "editable"-**false** key-value: "draft"-**false** String phoneNumber: the number to send message to int accountID: Account ID(0-5) String content: the message content to be sent
am command	am start -a android.intent.action.SENDTO -d smsto:phoneNumber --es sms_body content --ei account accountID --ez editable false --ez draft false
Return	Sending message

Save Message to Draft Box

Added in SDK version 1

Table 17: Save Message To Draftbox

Public Method	**Uri uri = Uri.parse("smsto:" + phoneNumber);
	Intent intent = new Intent(Intent.ACTION_SENDTO,uri);
	intent.putExtra("sms_body",content); intent.putExtra("editable",false);
	intent.putExtra("draft",true);
	intent.putExtra("account",int accountID); startActivity(intent);**

Description	To save message to draft box
Parameters	key-value: "editable"-**false** key-value: "draft"-**true** String phoneNumber: the number to send message to (optional) int accountID: Account ID(0-5)(Optional) String content: the message content to be sent
am command	am start -a android.intent.action.SENDTO -d smsto:phoneNumber --es sms_body content --ei account accountID --ez editable false --ez draft true
Return	Save the message to Draft box

Receive Message

Added in SDK version 1

Receiving messages via the following broadcasting message:

android:name="android.provider.Telephony.SMS_RECEIVED"/>

In the broadcasting message, there are three key-value pairs specified:

"number"-String value The sender's phone number
"content"-String value The message content
"account"-String value The account ID on the GXV3275 (from 0 to 5 for account 1 to account 6)

Table 18: Receive Message

Public Method	**private static String RECEIVE_MESSAGE=
	"android.provider.Telephony.SMS_RECEIVED"
	@Override
	public void onReceive(Context context,Intent intent){
	final String number = intent.getStringExtra("number"); final String content = intent.gerStringExtra("content"); final String account = intent.getStringExtra("account");
	}
	IntentFilter filter =new IntentFilter();
	filter.addAction(RECEIVE_MESSAGE);
	context.registerReceiver(myReceiver,filter);**

Description	To receive message
Parameters	key-value: "editable"-**false** key-valuse: "draft"-**true** String phoneNumber: the number to send message to (optional) int accountID: Account ID(0-5)(optional) String content: message content
am command	N/A
Return	Message

ACCOUNT API

Users could utilize the Account API to retrieve account ID and account name on **GXV3275**. The maximum number of accounts on GXV3275 is 6, with index from 0 to 5 for account 1 to account 6.

The following two classes are used in Account API:

- **com.base.module.account.AccountManager**
- **com.base.module.account.Account**

Firstly, an **AccountManager** instance is retrieved by **AccountManager.instance()**. Using this **AccountManager** instance, we can get **Account** instance. Then the account information can be retrieved via the methods in **Account** class.

Development Environment Setup

Added in SDK version 1

Before using the Account API, users need replace the **android.jar** file in the android-sdk-linux package with the one for GXV3275 (included in the GXV3275 SDK Package already). For example, in Android™ operating system 2.3, the **android.jar** file can be found in:

android-sdk-linux/platforms /android-10/android.jar

Replace this file with the one for GXV3275 in the GXV3275 SDK Package. And then refresh your project in Eclipse.

AccountManager CLASS API

Added in SDK version 1

AccountManager is used for managing Account class API. Firstly, import **AccountManager** class using the following code:

- **import com.base.module.account.AccountManager**

Then create an **AccountManager** instance by using method **AccountManager.instance()**. Now users could call other methods in **AccountManager** class.

AccountManager class has the following methods:

Table 19: AccountManager Interfaces

Public Method	AccountManager instance()
Description	Create an **AccountManager** instance
Parameters	N/A
am command	N/A
Return	An **AccountManager** instance
Public Method	**Account[ ] getAccounts(Context context)**
Description	Return all current accounts
Parameters	context
am command	N/A
Return	Account array
Public Method	**Account[ ] getActiveAccounts(Context context)**
Description	Return active accounts
Parameters	context
am command	N/A
Return	Active account array
Public Method	**Account getAccountByAccountID(Context context , int accountID)**
Description	Return Account according to account ID index
Parameters	Context, account ID (0 - 5)
am command	N/A
Return	Account
Public Method	**Account getAccountByOrderID(Context context , int accountID)**
Description	Return Account with actual display order
Parameters	Context, account ID (0 - 5)
am command	N/A
Return	Account
Public Method	**Account[ ] getActiveAccountsByOrder(Context context)**
Description	Return active account with actual display order
Parameters	context

am command	N/A
Return	Active account (ordered) array
Public Method	**Account[ ] getRegAccounts(Context context)**
Description	Return registered account
Parameters	context
am command	N/A
Return	Registered account array
Public Method	**Account[ ] getRegAccountsByOrder(Context context)**
Description	Return registered account with actual display order
Parameters	context
am command	N/A
Return	Registered account (ordered) array
Public Method	**void updateAccount(Context context,int accountID,Account account)**
Description	Update account according to Account ID
Parameters	Context, account ID (0 - 5), account number
am command	N/A
Return	N/A

Account CLASS API

Added in SDK version 1

Account class (**com.base.module.account.Account**) is the API to retrieve account information. Firstly, import the **Account** class using the following code:

- import com.base.module.account.Account

Then users could obtain the account information or modify account configuration using **Account** class.

Account class has the following methods:

Table 20: Account Interfaces

Public Method	**void setAccountID(int accountID)**
Description	Set Account ID
Parameters	Account ID (0 to 5)
am command	N/A
Return	N/A
Public Method	**void setAccountName(String name)**
Description	Set Account name
Parameters	Account name
am command	N/A
Return	N/A
Public Method	**void setSipServer(String serverPath)**
Description	Set SIP server address
Parameters	SIP server address
am command	N/A
Return	N/A
Public Method	**void setOutBoundProxy(String proxy)**
Description	Set outbound proxy address
Parameters	Outbound proxy address
am command	N/A
Return	N/A
Public Method	**void setSipUserID(String userID)**
Description	Set SIP user ID
Parameters	SIP user ID
am command	N/A
Return	N/A
Public Method	**void setSipAuthID(String AuthID)**
Description	Set SIP authorization ID
Parameters	SIP authorization ID
am command	N/A
Return	N/A

Public Method	void setSipAuthPassword(String password)
Description	Set SIP authorization password
Parameters	SIP authorization password
am command	N/A
Return	N/A
Public Method	void setVoiceMailUserID(String mailUserID)
Description	Set Voicemail user ID
Parameters	Voicemail user ID
am command	N/A
Return	N/A
Public Method	void setDisplayName(String displayName)
Description	Set SIP user display name
Parameters	SIP user display name
am command	N/A
Return	N/A
Public Method	void setActive(boolean active)
Description	Set account active status
Parameters	**true/false** for activate/deactivate
am command	N/A
Return	N/A
Public Method	void setRegistered(boolean reg)
Description	Set account registration status
Parameters	**true/false** for register/not register
am command	N/A
Return	N/A
Public Method	int getAccountID()
Description	Return Account ID
Parameters	N/A
am command	N/A
Return	Account ID

Public Method	String getAccountName()
Description	Return Account name
Parameters	N/A
am command	N/A
Return	Account name
Public Method	**String getSipServer()**
Description	Return SIP server address
Parameters	N/A
am command	N/A
Return	SIP server address
Public Method	**String getOutBoundProxy()**
Description	Return outbound proxy address
Parameters	N/A
am command	N/A
Return	Outbound proxy address
Public Method	**String getSipUserID()**
Description	Return SIP User ID
Parameters	N/A
am command	N/A
Return	SIP User ID
Public Method	**String getSipAuthID(Feature Request)**
Description	Return SIP authorization ID
Parameters	N/A
am command	N/A
Return	SIP authorization ID
Public Method	**String getSipAuthassword()**
Description	Return SIP authorization password
Parameters	N/A
am command	N/A
Return	SIP authorization password

Public Method	String getVoiceMailUserID()
Description	Return voicemail user ID
Parameters	N/A
am command	N/A
Return	Voicemail user ID
Public Method	String getDisplayName()
Description	Return SIP user display name
Parameters	N/A
am command	N/A
Return	SIP user display name
Public Method	boolean getActive()
Description	Return account active status
Parameters	N/A
am command	N/A
Return	**true/false** for activate/deactivate
Public Method	boolean getRegistered()
Description	Return account registration status
Parameters	N/A
am command	N/A
Return	**true/false** for register/unregister

CONTACT API

The Contact API in **GXV3275** SDK is inherited from Android™ operating system standard Contact API. Users could search the Contact database via the **Contact** class in **android.provider**. Also, **GXV3275** provides GS_ACCOUNT constant parameter in **ContactsContract.CommonDataKinds.Phone** class for SIP accounts on the phone. Users can directly call the **query** API, **insert** API, **update** API and **delete** API in Android **ContentRevolver** to operate on the Contact database.

Contact API Parameters

Added in SDK version 1

The following parameter represents the account ID (from 0 to 5 for account 1 to 6) for the SIP user.

- ContactsContract.CommonDataKinds.Phone.GS_ACCOUNT
  Description: **Type**: TEXT;
  Constant Value: "data11".

Retrieve Account ID of the Contact

Added in SDK version 1

Table 21: Retrieve Account ID of the Contact

Public Method	**Cursor phonesCursor = getContentResolver().query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null, ContactsContract.CommonDataKinds.Phone.CONTACT_ID + " = " + contactId, null, null); in t accountColumn = phonesCursor.getColumnIndex(Phone.GS_ACCOUNT); int accountID = phonesCursor.getInt(accountColumn);**
Description	To retrieve the account ID of the specified contact
Parameters	The cursor pointed to Contact database
am command	N/A
Return	The Account ID (from 0 to 5 for account 1 to 6) of the specified contact

Search Contact

Added in SDK version 1

Call ContentResolver class query (Uri uri, String[ ]projection, String selection, String[ ] selectionArgs, String sortOrder) in Android system to query database.

Table 22: Search Contact

Public Method	**Cursor query(Uri uri, String[ ] projection, String selection, String[ ] selectionArgs, String sortOrder);**
Description	To search contact information in Contact database by sending query with the provided URI
Parameters	• "**uri**": the URI to be retrieved, using the content:// scheme. • "**projection**": a list of columns to return. If it's null, all columns will be returned. • "**selection**": a filter specifying the rows to return. It's using the same format of SQL WHERE clause (not including "WHERE" itself). If it's null, all rows for the given URI will be returned. • "**selectionArgs**": if ?s is included in "selection", it will be replaced by the corresponding values from "selectionArgs", in the order specified in "selection". The values are strings. • **sortOrder**: Specify how to order the rows. It's using the same format of SQL ORDER BY clause (not including "ORDER BY" itself). If it's null, default order (sorted or unsorted) will be used.
am command	N/A
Return	A Cursor object at the beginning of the first matching entry; or null if no result is found

Update Contact Information

`Added in SDK version 1`

Call ContentResolver class update (Uri uri, ContentValues values, String where, String[ ] selectionArgs) in Android system to update database.

Table 23: Update Contact Information

Public Method	**int update(Uri uri, ContentValues values, String where, String[ ] selectionArgs);**
Description	To update contact information by updating row(s) in a content URI
Parameters	• "**uri**": the URL to be modified. • "**values**": the new field values. The key is the column name of the field. If it's null, the existing field value will be removed. • "**where**": a filter specifying the rows to be updated. It's using the same format of SQL WHERE clause (not including "WHERE" itself). If it's null, all rows for the given URI will be returned. • "**selectionArgs**": if ?s is included in "selection", it will be replaced by the corresponding values from "selectionArgs", in the order specified in "selection". The values are strings.

am command	N/A
Return	The number of rows updated
Throw	NullPointerException will be thrown if "uri" or "values" parameter is null

Add Contact Information

Added in SDK version 1

Call ContentResolver class Uri insert (Uri uri, ContentValues values) in Android system to insert contacts to database.

Table 24: Add Contact Information

Public Method	**Uri insert(Uri uri, ContentValues values);**
Description	To add contact information by inserting a row into a table at the given URI
Parameters	• "**uri**": the URL of the table to insert the contact into. • "**values**": the values for the inserted row. The key is the column name of the field. If it's null, an empty row will be created.
am command	N/A
Return	The URL of the newly created row

Delete Contact Information

Added in SDK version 1

Call ContentResolver class int delete (Uri uri, String where, String[ ] selectionArgs) in Android system to delete contacts from database.

Table 25: Delete Contact Information

Public Method	**int delete (Uri uri, String where, String[ ] selectionArgs);**
Description	To delete contact information by specifying a content URI
Parameters	• "**uri**": the URL of the row to be deleted. • "**where**": A filter to speficify the rows to be deleted. It's using the same format of SQL WHERE clause (not including "WHERE" itself). • "**selectionArgs**": if ?s is included in "selection", it will be replaced by the corresponding values from "selectionArgs", in the order specified in "selection". The values are strings.
am command	N/A
Return	The number of rows deleted

CALL LOG API

The CallLog API in GXV3275 SDK is inherited from Android™ operating system standard CallLog API. Users could search the CallLog database via **CallLog Provide**. Additionally, GXV3275 provides more **GS_ACCOUNT** constant parameter for SIP accounts on the phone.

Call Log API Parameters

Added in SDK version 1

The following parameter represents the account ID (from 0 to 5 for account 1 to 6) for the SIP user.

- CallLog.Calls.GS_ACCOUNT
  Description: **Type**: TEXT;
  Constant Value: "account".

Retrieve Account ID of The CallLog Entry

Added in SDK version 1

Table 26: CallLog API Usage

Public Method	**Cursor cursor = cr.query(CallLog.Calls.CONTENT_URI, null, null,null, CallLog.Calls.DEFAULT_SORT_ORDER);** **String gsAccount = cursor.getString(cursor .getColumnIndex(CallLog.Calls.GS_ACCOUNT));**
Description	To retrieve the account ID of the specified call log entry
Parameters	The cursor pointed to CallLog database
am command	N/A
Return	The Account ID (from 0 to 5 for account 1 to 6) of the specified call log entry

AUDIO CHANNEL API

Added in SDK version 1

GXV3275 supports the audio channel API for handset, speaker and headset (wired). Headset and speakerphone API are provided by Android while Handset API is added from GXV3275.

The methods listed in this section are provided by **android.media.AudioManager** class and they can be used to search or configure audio channel. Before using the listed methods, please obtain the instance of the class first using **Context.getSystemService(Context.AUDIO_SERVICE)**.

Retrieve Channel Type

Table 27: Retrieve Channel Type

Public Method	**boolean isHandsetOn()**
Description	To retrieve the channel status for handset
Parameters	N/A
am command	N/A
Return	**true/false** for valid/invalid handset channel
Public Method	**boolean isWiredHeadsetOn()**
Description	To retrieve the channel status for headset (wired)
Parameters	N/A
am command	N/A
Return	**true/false** for valid/invalid headset channel
Public Method	**boolean isSpeakerphoneOn()**
Description	To retrieve the channel status for speakerphone
Parameters	N/A
am command	N/A
Return	**true/false** for valid/invalid speakerphone channel

Configure Channel Type

Firstly, add the privilege to modify the media configuration in **AndroidManifest.xml** as follows:

```
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
```

Table 28: Configure Channel Type

Public Method	**void setHandsetOn(boolean on)**
Description	To configure the channel for handset
Parameters	• "**true**": turn on the channel for handset. • "**false**": turn off the channel for handset and switch to speakerphone.
am command	N/A
Return	N/A
Public Method	**void setSpeakerphoneOn(boolean on)**
Description	To configure the channel for speakerphone
Parameters	• "**true**": turn on the channel for speakerphone. • "**false**": turn off the channel for speakerphone and switch to handset.
am command	N/A
Return	N/A
Public Method	**void setWiredHeadsetOn(boolean on)**
Description	To configure the channel for headset
Parameters	• "**true**": turn on the channel for headset (wired). • "**false**": turn off the channel for headset (wired) and switch to speakerphone.
am command	N/A
Return	N/A

HARD KEYS API

Added in SDK version 1

GXV3275 supports hard keys API for users to detect the key pressing events in the development. The value of the keys is stored in **KeyEvent** class.

The following table shows the key listener event API provided for the hard keys on GXV3275Key:

Table 29: Hard Keys API

Key Listener API	Description
public static final int KEYCODE_PHONEBOOK = 200;	PHONEBOOK Key
public static final int KEYCODE_HOLD = 201;	HOLD Key
public static final int KEYCODE_HEADSET = 202;	HEADSET Key
public static final int KEYCODE_MSG = 203;	MESSAGE Key
public static final int KEYCODE_TRNF = 204;	TRANSFER Key
public static final int KEYCODE_CONF = 205;	CONFERENCE Key
public static final int KEYCODE_SEND = 206;	SEND Key
public static final int KEYCODE_SPEAKER = 207;	SPEAKER Key

OTHER API

Restart Provision

`Added in SDK version 6`

GS Android Phone provides system operation API. Please refer to the below:
Class: **com.base.module.system.SystemManager**.

Developers can use **Context.getSystemService(Context.SYSTEM_MANAGER_SERVICE)** to get an instance.

Table 30: System Operation API

Public Method	void restartProvision()
Description	Restart the system firmware update checking. Different from the check after reboot.
Parameters	N/A
am command	N/A
Return	N/A

LED Control API

`Added in SDK version 7`

GS Android Phone supports LED Control API for users to turn on or off LED signal light at up-right of the device. **LightsManager** is used for managing LED class API.

Firstly, import **LightsManager** class using the following code:

- **import android.hardware.LightsManager**

Then create a **LightsManager** instance by using method

- **LightsManager mLightsMannager = (LightsManager)getSystemService(Context.LIGHTS_SERVICE);**

LightsManager class has the following methods:

Table 31: LED Control API

Public Method	int startLedLight(int level, int lightColor, int onMs, int offMs)
Description	To turn on the LED light
Parameters	• level: (0 - 6) priority of thr lights from low to high • lightColor : lights color(red and green) LightsManager.COLOR_GREEN(0x0000FF00) LightsManager.COLOR_RED(0x00FF0000) • onMs:lights on milliseconds • offMs:lights off milliseconds (1000,0) the LED is on (0,1000) the LED is off (500,500) the LED is flashing

am command	N/A
Return	int: index to turn off the LED light
Public Method	**void closeLight(int index)**
Description	To turn off the LED lights
Parameters	• index: the return value from startLedLight method
am command	N/A
Return	N/A

Note: For all the LED light turned on by calling **startLedLight,** please call the method **closeLight** to turn off them before the program ends.

```
mLightsMan = (LightsManager)getSystemService(Context.LIGHTS_SERVICE);
int indexGreen = mLightsMan.startLedLight(1, 0x0000FF00 ,1000, 0);
int indexRed = mLightsMan.startLedLight(2, 0x00FF0000 ,1000, 500);
mLightsMan.closeLight(indexGreen);
mLightsMan.closeLight(indexRed);
```

ADB COMMANDS

Added in SDK version 1

GXV3275 supports the ADB commands introduced in this section. Developers could use these commands for debugging purpose.

Connect/Disconnect Commands

- **connect <host>[:<port>]**
  Description: Connect to a device via TCP/IP;
  Port 5555 is used by default if no port number is specified.

- **disconnect [<host>[:<port>]]**
  Description: Disconnect from a TCP/IP device;
  Port 5555 is used by default if no port number is specified. Using this command with no additional arguments will disconnect from all connected TCP/IP devices.

Device Commands

- **adb push <local> <remote>**
  Description: Copy file/dir to device.

- **adb pull <remote> [<local>]**
  Description: Copy file/dir from device.

- **adb logcat [<filter-spec>]**
  Description: View device log.

- **adb install [-l] [-r] [-s] <file>**
  Description: Push this package file to the device and install it.
  '-l': forward-lock the app
  '-r': reinstall the app, keeping its data
  '-s': install on SD card instead of internal storage

- **adb uninstall [-k] <package>**
  Description: Remove this app package from the device.
  '-k': keep the data and cache directories

- **adb help**
  Description: Show this help message.

- **adb version**
  Description: Show version num.

Scripting

- **adb wait-for-device**
  Description: Block until device is online.

- **adb start-server**
  Description: Ensure that there is a server running.

- **adb kill-server**
  Description: Kill the server if it is running.

gs.jar USAGE

Added in SDK version 1

gs.jar is required in compilation. Please import this library to the project and put it in the most front place.