Grandstream Networks, Inc.

GXP2200

Framework Service Guide V1.0

Grandstream Networks, Inc.

www.grandstream.com

Index

OVERVIEW

Grandstream GXP2200 Enterprise Multimedia Phone for Android™ is an innovative smart desk phone loaded with tremendous value for web-integrated business communications as well as capability for advanced custom applications development and personalization. Based on Android™ operating system 2.3, GXP2200 is compatible with most of the standard Android™ operating system 2.3 API except the phone call and message part as described in this document.

The Android™ operating system API information can be found in the following link for your reference:
http://developer.android.com/develop/index.html

The following system functions are provided in GXP2200 SDK for user's development and customization:

- The 3rd party application can use system functions to dial number and make calls.
- The 3rd party application can use system functions to send messages.

This GXP2200 Framework Service Guide describes the details of interface for dialing/calling and sending messages.

DIAL INTERFACE

DIAL module interface receives incoming broadcasting signals. The corresponding action in broadcasting is "**com.base.module.phone.DIAL**".

DIAL INTERFACE ACTIONS

- **intent.putExtra("action", "dialcheck");**
  Description: Edit number before dialing out.

- **intent.putExtra("action", "dialout");**
  Description: Dial out the number directly.

- **intent.putExtra("action", "dialrecent");**
  Description: Redial the last dialed number.

DIAL INTERFACE PARAMETERS

- **intent.putExtra("account", accountID);**
  Description: To associate with phone's account ID.

- **intent.putExtra("number", number);**
  Description: The number to be dialed out.

- **intent.putExtra("inviteType", isVideo);**
  Description: To make calls with video or audio only.

DIAL INTERFACE FUNCTIONS

FUNCTION 1: OPEN DIAL PANEL

Description	To open the dial panel
Usage	• Android interface: **Intent intent = new Intent("com.base.module.phone.DIAL");** **intent.putExtra("action", "dialcheck");** **sendBroadcast(intent);**

	• Command: **am broadcast -a com.base.module.phone.DIAL -e "action" "dialcheck"**
Parameter Type	• Parameter: "action" Type: string

FUNCTION 2: EDIT NUMBER BEFORE DIALING

Description	To edit number before dialing out
Usage	• Android interface: **Intent intent = new Intent("com.base.module.phone.DIAL");** **intent.putExtra("action", "dialcheck");** **intent.putExtra("account", accountID);** **//Associate with this account ID** **intent.putExtra("number", number) ;** **//Edit this number before dialing** **sendBroadcast(intent);** • Command: The following command example uses account 1 to dial 3301. **am broadcast -a com.base.module.phone.DIAL --ei "account" 0 -e "number" "3301" -e "action" "dialcheck"**
Parameter Type	• Parameter: "action" Type: string • Parameter: "account" Type: int • Parameter: "number" Type: string

FUNCTION 3: DIAL OUT

Description	To dial out
Usage	• Android interface: **Intent intent = new Intent("com.base.module.phone.DIAL");** **intent.putExtra("action", "dialout");** **intent.putExtra("account", accountID);** **//Associate with this account ID** **intent.putExtra("number", number);** **//Edit this number before dialing** **intent.putExtra("inviteType", isVideo);** **//Video call or audio call only**

	sendBroadcast(intent); • Command: The following command example uses account 1 to dial 3301 to make an audio call. **am broadcast -a com.base.module.phone.DIAL --ei "account" 0 -e "number" "3301" --ei "inviteType" 0 -e "action" "dialout"**
Parameter Type	• Parameter: "action" Type: string • Parameter: "account" Type: int • Parameter: "number" Type: string (Direct IP call is supported) • Parameter: "inviteType" Type: int Possible values: 0 - audio call only; 1 - video call

FUNCTION 4: REDIAL

Description	To redial the last dialed number
Usage	• Android interface: **Intent intent = new Intent("com.base.module.phone.DIAL");** **intent.putExtra("action", "dialrecent");** **sendBroadcast(intent);** • Command: **am broadcast -a com.base.module.phone.DIAL -e "action" "dialrecent"**
Parameter Type	• Parameter: "action" Type: string

MESSAGE INTERFACE

EDIT MESSAGE BEFORE SENDING

ACTIONS

- **android.intent.action.SEND**
- **com.base.module.message.send**

EXAMPLE

```
Intent sendIntent = new Intent("com.base.module.message.send");
//The only way to send out message directly. This will direct to the sending message panel.
```

Or, users could use:

```
Intent sendIntent = new Intent(Intent.ACTION_SEND);
//There are several ways to send out message (e.g., Bluetooth, Email or other programs).
//Users could choose the sending program to be used.

sendIntent.setType("text/plain");
//Set text type. Mandatory.

endIntent.putExtra(Intent.EXTRA_TEXT, "");
//Message content. Optional. Users can add it in message editing screen.

sendIntent.putExtra("number", phoneNumber);
//The number to send the message to. Optional. Users can add it in message editing screen.

sendIntent.putExtra("accountId", accountId);
//Account index 0 - 5 for account 1 - 6 respectively. Optional. Default value is 0, i.e., account 1.

sendIntent.putExtra("localcontact",false);
//Whether the contact in local phonebook or not. Optional. Default value is true.

context.startActivity(sendIntent);
```

DATABASE EDIT/DELETE/SEARCH/UPDATE/PROVIDER INTERFACE

TABLE AND FIELD FOR INBOX/OUTBOX

```
public static class Message{
        public static final String TB_NAME = "message";
        //Table name.

        public static final String ID = "id";
        //Primary Key ID.

        public static final String ACCOUNT = "account";
        //Account ID (e.g., 3205). Mandatory.

        public static final String CONTENT = "content";
        //Content of the last message.

        public static final String LAST_DETAIL_ID = "detailid";
        //primary key ID for the last message in table "messagedetail".

        public static final String CREATTIME = "creattime";
        //Create time.

        public static final String UNREAD_DETAIL_COUNT = "unreaddetailcount";
        //Number of unread messages in table "messagedetail".

        public static final String TOTAL_DETAIL_COUNT = "detailtotalcount";
        //Total number of messages in messagedetail table.

        public static final String CONTACT_NAME = "contactname";
        //Contact name.

        public static final String CONTACT_NUMBER = "contactnumber";
        //Contact number (e.g., 3206). Mandatory.

        public static final String CONTACT_PHOTO = "contactphoto";
        //Contact picture
}

public static class Message_detail{
        public static final String TB_NAME = "messagedetail";
```

```
        public static final String ID = "id";
        public static final String CONTACT_NUMBER = "contactnumber";
        public static final String ACCOUNT = "account";
        public static final String CONTENT = "content";
        public static final String CREAT_TIME = "sendtime";
        public static final String SEND_STATE = "sendstate";
        //Sending status. 0: sending; -1: sent successfully; 4: sent failed.

        public static final String READ_STATE = "readstate";
        //Currently not used.

        public static final String TYPE = "type";
        //Message type. 0: sent; 1: received.

        public static final String CLOCK = "clock";
        //Message locked/unlocked. 1: locked; -1: unlocked.
}
```

NOTE:

The relationship between table "message" and table "messagedetail" is one-to-many. The same message between Contacts and account is one entry in table "message". However, in table "messagedetail", there might be more than one. Contacts and account might have sent messages multiple times, which are related by account ID and contact number.

TABLE AND FIELD FOR DRAFT BOX

```
public static class Draft{
        public static final String TB_NAME = "draft";
        public static final String ID = "id";
        public static final String CONTACT_NUMBER = "contactnumber";
        //Invalid field.

        public static final String ACCOUNT = "account";
        //Invalid field.

        public static final String CONTACT = "contact";
        //One draft message might have several contacts saved to send the message to. In this
        //case, combine the string in "contact number-account ID" format. For example:
        //"3256-32051;32956-32051", where 3256 and 32956 are the contacts to receive the
        //message, 32051 is the account ID.
```

```
        public static final String CONTENT = "CONTENT";
        public static final String CREAT_TIME = "creattime";
        public static final String STATE = "state";
        //Currently not used.
}

public static class Draft_detail{
        public static final String TB_NAME = "draftdetail";
        public static final String ID = "id";
        public static final String ACCOUNT = "account";
        public static final String CONTACT_NAME = "contactname";
        public static final String CONTACT_NUMBER = "contactnumber";
        public static final String CONTACT_PHOTO = "contactphoto";
        public static final String DRAFT_ID = "draftid";
        //Foreign key. Mandatory.
}
```

NOTE:

The relationship between table "draft" and table "draft_detail" is one-to-many.

PROVIDER INTERFACE

Users could use **ContentResolver** to add/delete/update/insert table according to the uri.

```
uri: AUTHORITY = "com.base.module.message";

Uri.withAppendedPath(Uri.parse("content://" + AUTHORITY + ""), "message");
//Operate on table "message".

Uri.withAppendedPath(Uri.parse("content://" + AUTHORITY + ""), "message/#");
//Operate on table "message". Search condition based on primary ID.

Uri.withAppendedPath(Uri.parse("content://" + AUTHORITY + ""), "messagedetail");
//Operate on table "messagedetail".

Uri.withAppendedPath(Uri.parse("content://" + AUTHORITY + ""), "messagedetail/#");
//Operate on table "messagedetail". Search condition based on ID.
```

Uri.withAppendedPath(Uri.parse("content://" + AUTHORITY + ""), "draft");
//Operate on table "draft".

Uri.withAppendedPath(Uri.parse("content://" + AUTHORITY + ""), "draft/#");
//Operate on table "draft". Search condition based on ID.

Uri.withAppendedPath(Uri.parse("content://" + AUTHORITY + ""), "draftdetail");
//Operate on table "draftdetail".

Uri.withAppendedPath(Uri.parse("content://" + AUTHORITY + ""), "draftdetail/#");
//Operate on table "draftdetail". Search condition based on ID.

HOW TO SEND OUT A MESSAGE

To send out a message, please follow the steps below:

- Insert the message in database.
  Please insert the message in both table "message" and table "messagedetail". When inserting in table "messagedetail", set "type" to "0" and "sendstate" to "0".

- Send out broadcasting.
  action="com.base.module.sms.action.send_broadcast"

- Wait for about 1 to 5 seconds.

- Check the sendstate value.
  Check the sendstate value. If sendstate=-1, the message is sent out successfully. If sendstate=4, the sending out is failed.

NOTE:

- Once the message is sent out, the Draft box message needs to be deleted.
- Creattime is measured in ms, using the following method:
  new Date().getLongTime