202006051540.3.0.2

1.MNOpenSDK集成（MNOpenSDK integration）

1. 将mn_sdk.aar导入到项目lib目录下。（Import mn_sdk.aar into the project lib directory.）
2. 在app的build.gradle中加入SDK依赖（Add SDK dependencies in app's build.gradle）

```
dependencies {
    implementation fileTree(include: ['*.jar'], dir: 'libs')
    //compile project(path: ':mn_sdk')
    implementation(name: 'mnopensdk-3.0', ext: 'aar')
    ......
    ......
}
```

3. 在Application中进行SDK初始化工作（SDK initialization in Application）

```
public class BaseApplication extends MNApplication {
    public static String APP_KEY = "31cc93923faa4bff";// 公共KEY
    public static String APP_SECRET ="f6c9deec31644885a123c8ffb573a52e";
    public static String HOST_DOMAIN = "https://restcn.bullyun.com";

    @Override
    public void onCreate() {
        super.onCreate();
        // 初始化MNSDK(Initialize MNDSK)
        MNOpenSDK.initWithKeyAndSecret(this, APP_KEY, APP_SECRET);
        /**
        *设置域名,可以在需要切换域名的时候进行设置(Set the domain name,
        *you can set it when you need to switch the domain)
        *(Default domain:https://restcn.bullyun.com)
        */
        MNOpenSDK.setMNKitDomain(this, HOST_DOMAIN);
        /** 设置是否需要提前建立与设备P2P连接关系（Set whether to
        * establish a P2P connection with the device in advance）
        */
        MNOpenSDK.setP2pPreLinkState(this, true);
    }

    // 服务透传消息，用户可以根据自己需求去实现
    @Override
    public void OnServerMSG(ServerMsgBean data) {
    /**
    * 1：设备分享（Device sharing）
    * 2：取消设备分享或者设备分享过期（Cancel device sharing or device sharing
expired）
    * 3：添加设备（Add device）
    * 4：解绑设备（Unbundling device）
    * 5：设备上线（Device goes online）
    * 6：设备下线（Device offline）
    * 7：用户密码修改（User password modification）
```

```
    * 8：低功耗设备休眠（Low-power device sleep）
    * 10：被分享设备权限变更（Shared device permissions change）
    */
        myHandler.post(() -> {
            if (data.getActionType() == 1) {
                // 他人分享设备给本账号了
            } else if (data.getActionType() == 2) {
                // 主账号取消设备分享或者设备分享过期
            } else if (data.getActionType() == 3) {
                // 添加设备
            } else if (data.getActionType() == 4) {
                // 解绑设备
            } else if (data.getActionType() == 5) {
                // 有设备上线啦
            } else if (data.getActionType() == 6) {
                // 有设备离线啦
            } else if (data.getActionType() == 7) {
                // 7：用户密码修改
            } else if (data.getActionType() == 8) {
                // 8：低功耗设备休眠
            } else if (data.getActionType() == 10) {
                // 10：被分享设备权限变更
            }
        });
    }
}
```

4. 登录（log in）

```
private void Login(String userName, String password) {
    MNKit.loginWithAccount(userName, password, new
MNKitInterface.LoginCallBack() {
        @Override
        public void onLoginFailed(String s) {
            if (progressHUD != null) {
                progressHUD.dismiss();
            }
            ToastUtils.MyToastCenter(getString(R.string.net_err_and_try));
        }

        @Override
        public void onLoginSuccess(LoginBean loginBean) {
            if (progressHUD != null) {
                progressHUD.dismiss();
            }
            SharedPreferUtils.write(TAG, "username", userName);
            SharedPreferUtils.write(TAG, "password", password);
            startActivity(newIntent(LoginActivity.this, HomeActivity.class));
            finish();
        }
    });
}
```

5. 注销与退出登录（Logout and logout）

```
//按 物理返回键
```

```
long _lLastBack = 0;
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_BACK) {
        long cur = System.currentTimeMillis();
        if (cur - _lLastBack < 1500) {
            threadPool.execute(() -> {
                // 注销账号或者退出APP（Cancel account or exit APP）
                MNOpenSDK.logout();
                mainHandler.post(() -> {
                    MNApplication.getInstance().mActivityStack.AppExit();
                });
            });
        } else {
            _lLastBack = cur;
            ToastUtils.MyToastBottom("再按一次，退出应用");
        }
        return true;
    } else {
        return super.onKeyDown(keyCode, event);
    }
}
```

2.设备绑定（Device binding)

1.无线设备绑定（Wireless device binding)

2.1.1. 设置配网结果监听器（Set up network configuration results listener)

```
MNBindDevProcessor.getInstance().register(this);
/**
* WIFI网络配置监听,当设备网络配置成功并且上线之后，会回调此方法
* WIFI network configuration monitoring, this method will be called
* back when the device network configuration is successful and online。
* @param pszJson Example:
* {
*   "deviceSn":"MDAhAQEAbGUwNjFiMjIzOWMxNgAA",
*   "vn":"ABCDEF",
*   "bindState:":3002
* }
* bindState:3000或null 设备没有被绑定，调用绑定接口进行绑定
* (The device is not bound, call the binding interface to bind);
* bindState:3001 设备被其他用户绑定，可以申请解绑
* (The device is bound by another user and can apply for unbinding);
* bindState:3002 设备已经被自己绑定，提示配网成功，直接查看设备
* (The device has been bound by itself, prompting that the
* network configuration is successful, check the device directly);
* @param nlen
*/
@Override
public void OnRequestToBindDevice(String pszJson, int nlen) {
    . . . . . .
}
```

2.1.2. 设置声波配网监听器（Setting up a sonic distribution network monitor)

```
SoundWaveManager.getIntance().setListener(new VoicePlayerListener() {
//开始发送声波回调，可以在这里设置发送声波时的UI变化。
//Start sending sonic callback. Here you can set UI changes
//when sending sonic.
@Override
public void onPlayStart(VoicePlayer voicePlayer) {
}
    //声波发送完成，可以在这里设置声波发送完成时的UI变化。
    //Acoustic wave transmission is completed. You can set
    //the UIchanges when the acoustic wave transmission is completed.
@Override
public void onPlayEnd(VoicePlayer voicePlayer) {

}
});
```

2.1.3. 配置网络，发送声波（Configure the network to send sound waves)

```
/**
* 将WIFI-SSID与密码通过声波方式发送给设备
* (Send WIFI-SSID and password to the device by sound wave)
* @param ssid(String) WIFI-SSID
* @param password(String) WIFI-password
*/

SoundWaveManager.getIntance().sendMsg(ssid, password);
```

2.1.4. 绑定设备（Bind device）

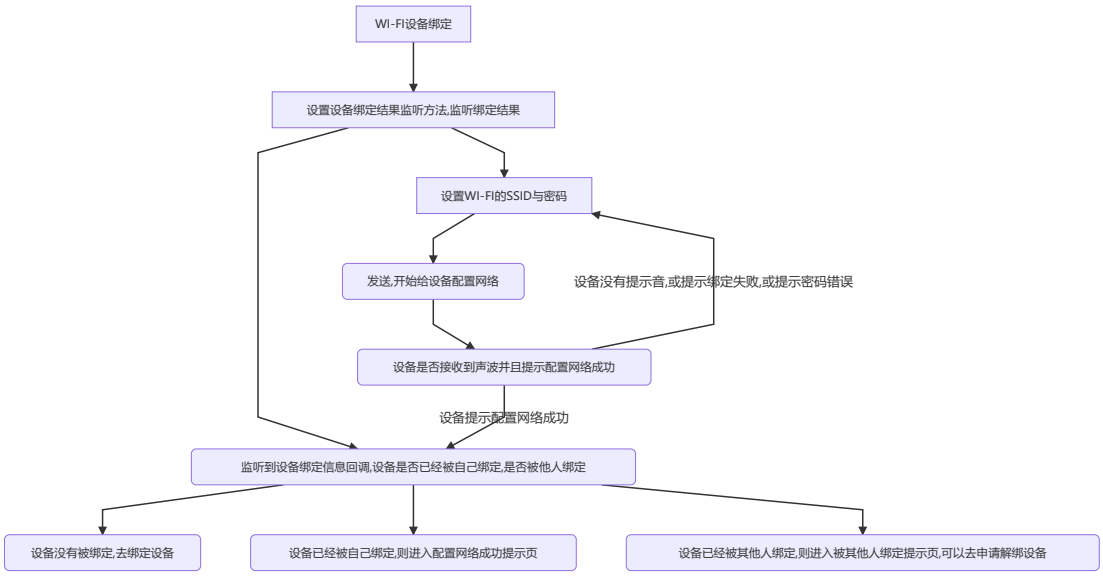
```
/**
* 发送完声波之后，等待设备接收声波信息，当设备配置网络成功之后，会主动回调此方法
* (After sending the sound wave, wait for the device to receive the >sound
* wave information. After the device successfully configures the >network,
* it will actively call back this method)
*/
@Override
public void OnRequestToBindDevice(String pszJson, int nlen) {
    if(pszJson == null){
        return;
    }
    BindDevBeean bindDevBeean = new Gson().fromJson(pszJson,
>BindDevBeean.class);
   if (bindDevBeean == null || bindDevBeean.getDeviceSn()==null){
       return;
   }
   String deviceSn = bindDevBeean.getDeviceSn();
   String vn = bindDevBeean.getVn();
   int bindState = bindDevBeean.getBindState();
    if(vn == null){
       vn = "ABCDEF";
   }
   if (bindState == 0 || bindState == 3000) {
       // 设备没有被绑定，去绑定设备(The device is not bound,
       // go to bind the device)
      MNKit.bindDeviceBySnAndVn(deviceSn, vn, this);
   } else if(bindState == 3001){
      // 设备已被其他用户绑定,可以申请解绑设备(The device has been bound by
```

```
        // another user, you can apply for unbinding the device)
    }else if(bindState == 3002){
        // 设备已经被自己绑定,可以直接查看设备啦(The device has been
        // bound by itself, you can view the device directly)
    }
}
```

2.1.5. 移除监听器(Remove listener)

```
/**
* 配置完成或者主动退出时，移除监听器
* (Remove the listener when the configuration is
* complete or actively exit)
*/
@Override
protected void onDestroy() {
    super.onDestroy();
    SoundWaveManager.getIntance().destroy();
    MNBindDevProcessor.getInstance().unregister(this);
    MNApplication.getInstance().mActivityStack.removeActivity(this);
}
```

6. WIFI 设备绑定流程图



2.有线设备绑定(Wired device binding)

2.2.1.获取设备SN(Get Device SN)

```
//通过扫码设备二维码获取其他方式获取到设备的SN
//btain the SN of the device by scanning the QR code of the device
deviceSn = getIntent().getStringExtra("deviceSn");
```

2.2.2. 有线配网与绑定设备(Wired network and binding device)

```
if (!TextUtils.isEmpty(deviceSn)) {
// 已经获取到SN，尝试有线绑定（Already obtained SN, try to wire bind）
    WiredBindingManager.getIntance().startWiredBinding(deviceSn, new
WiredBindingManager.WiredBindingCallBack() {
        @Override
```

```
        public void onWiredBindingSuc(String sn) {
        // 设备有连接上网络，开始绑定设备到账号(The device is connected to
        // the network and begins to bind the device to the account)
            MNKit.bindDeviceBySnAndVn(deviceSn,vn,listener);
        }

        @Override
        public void onWiredBindingFailed() {
        // 设备没有有连接上网络，有线绑定失败（The device is not connected
        // to the network, and the wire binding fails）
        }
    });
}
```

3.直播(Live)

3.1. 实现播放器（Implement the player)

```
<RelativeLayout
    android:id="@+id/rl_main_lay"
    android:layout_width="match_parent"
    android:layout_height="220dp">

    <com.mn.player.MNPlayControl
        android:id="@+id/mn_play_control"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    </com.mn.player.MNPlayControl>
</RelativeLayout>
```

```
//在播放器Activity中
mnPlayControl = findViewById(R.id.mn_play_control);
```

3.2. 设置播放器监听（Set the player to listen)

```
private void initLinstener() {
    mnPlayControl.setPlayControlLinstener(new MNPlayControlLinstener() {
        @Override
        public void OnTaskStatus(int videoType, int taskStatus, float fProgress)
{
        LogUtil.i(TAG, "OnTaskStatus videoType : " + videoType);
            if (videoType == MTT_P2P_REALPLAY.ordinal()) {
                // 直播任务
                if (taskStatus == MNTS_READY.ordinal()) {
                    Log.i(TAG, "OnTaskStatus -- 准备就绪...");
                } else if (taskStatus == MNTS_RUNNING.ordinal()) {
                    Log.i(TAG, "OnTaskStatus -- 播放中..." + fProgress);
                } else if (taskStatus == MNTS_DEVICE_OFFLINE.ordinal()) {
                    Log.i(TAG, "OnTaskStatus -- 设备不在线...");
                } else if (taskStatus == MNTS_CONNECT_FAILED.ordinal()) {
                    Log.i(TAG, "OnTaskStatus -- 连接失败，请重试...");
                } else if (taskStatus == MNTS_STOPPED.ordinal()) {
                    Log.i(TAG, "OnTaskStatus -- 结束");
```

```
                } else if (taskStatus == MNTS_DESTROYED.ordinal()) {
                    Log.i(TAG, "OnTaskStatus -- 任务已销毁...");
                } else if (taskStatus == MNTS_CANCELING.ordinal()) {
                    Log.i(TAG, "OnTaskStatus -- 任务正在取消...");
                } else if (taskStatus == MNTS_CONNECTING.ordinal()) {
                    Log.i(TAG, "OnTaskStatus -- 正在获取视频...");
                } else if (taskStatus == MNTS_PAUSED.ordinal()) {
                    Log.i(TAG, "OnTaskStatus -- 暂停");
                } else {
                    Log.i(TAG, "OnTaskStatus -- 其他任务类型");
                }
            } else if (videoType == MTT_P2P_SD_PLAYBACK.ordinal()) {
                // 卡回放
            } else if (videoType == MTT_CLOUD_ALARM.ordinal()) {
                // 云回放
            }
        }

        @Override
        public void onDownloadTaskStatus(boolean taskStatus, int downStatus,
float progress, String fileName) {
            runOnUiThread(() -> {
                if (taskStatus) {
                    if (downStatus == 0 || downStatus == 1) {
                        // 开始任务
                        LogUtil.i(TAG, "下载中... " + progress);
                    } else if (downStatus == 2) { // 下载失败
                        LogUtil.i(TAG, "下载失败");
                    } else if (downStatus == 3) { // 下载成功
                        LogUtil.i(TAG, "下载成功 :  " + fileName);
                    } else if (downStatus == 4) {
                        LogUtil.i(TAG, "取消下载 :  ");
                    }
                } else {
                    LogUtil.i(TAG, "下载失败");
                    tvProgress.setText("失败");
                }
            });
        }

        @Override
        public void onSetSessionCtrl(boolean result, String pointId) {
            // 设置预置点结果回调
            if (result) {
                if (loadingDialog != null) {
                    loadingDialog.show();
                }
                MNKit.getFavoritePointsInfo(mDevice.getId(),
MNPlayControlActivity.this);
            } else {
                if (loadingDialog != null) {
                    loadingDialog.dismiss();
                }
                LogUtil.i(TAG, getString(R.string.tv_failed_set_favorites));

 ToastUtils.MyToastBottom(getString(R.string.tv_failed_set_favorites));
            }
        }
```

```
        @Override
        public void onDelSessionCtrl(boolean result, String pointIndex) {
            //删除摇头机预置点
            if (result) {
                MNKit.getFavoritePointsInfo(mDevice.getId(),
MNPlayControlActivity.this);
            } else {
                if (loadingDialog != null) {
                    loadingDialog.dismiss();
                }
                ToastUtils.MyToastBottom(getString(R.string.net_err_and_try));
            }
        }

        @Override
        public void onGotoSessionCtrl(boolean result, String pointIndex) {
            // 前往摇头机预置点
            ToastUtils.MyToastBottom(getString(R.string.goto_preset_point));
        }

        @Override
        public void runSpeed(int nYear, int nMonth, int nDay, int nHour, int
nMinute, int nSecond, long lNetworkFlowPerSecond) {
            LogUtil.i(TAG, String.format("%d-%d-%d %d:%d:%d    ↓%.2fkb/s",
nYear, nMonth, nDay, nHour, nMinute, nSecond, (float) lNetworkFlowPerSecond /
1024));
        }

        @Override
        public void onAudioSwitchChanged(boolean open) {
            if (open) {
                btnVoice.setTag("on");
                btnVoice.setImageResource(R.mipmap.live_list_btn_sound_pre);
            } else {
                btnVoice.setTag("off");
                btnVoice.setImageResource(R.mipmap.live_list_btn_sound);
            }
        }

        @Override
        public void onHoldTalkSwitchChanged(boolean open) {
            if (open) {

 btnSingleTalk.setImageResource(R.mipmap.live_list_btn_voice_pre);
            } else {
                btnSingleTalk.setImageResource(R.mipmap.live_list_btn_voice);
            }
        }

        @Override
        public void onToborder() {
            LogUtil.i(TAG, "转动到边界啦");
            ToastUtils.MyToastBottom(getString(R.string.tv_goto_the_border));
        }

        @Override
        public void onNvrSessionView(String data) {
```

```
        }

        @Override
        public void onAudioVolume(int volume) {

        }
    });
}
```

3.3. 设置数据，开始直播任务（Set the data and start the live broadcast task)

```
    mnPlayControl.setDeviceInfo(mDevice);
    mnPlayControl.startLive(MNControlAction.MNCodeStream.VIDEO_FUL, 0);
```

3.4.播放器暂停，重新开始与销毁（Pause, restart and destroy the player)

```
@Override
protected void onResume() {
    super.onResume();
    //恢复播放任务
    if (!isFristInto) {
        if (mnPlayControl.getTaskStatus() == MNTS_PAUSED.ordinal()) {
            mnPlayControl.resumePlayer();
        } else {
            // 这里重新开启直播代码
            initData();
        }
    }
    isFristInto = false;
}

@Override
protected void onPause() {
    super.onPause();
    // 暂停播放任务
    if (!isFinish) {
        if (mnPlayControl.getTaskStatus() == MNTS_RUNNING.ordinal()) {
            mnPlayControl.pausePlayer();
        } else {
            mnPlayControl.stopPlayer();
        }
    }
}

@Override
protected void onDestroy() {
    super.onDestroy();
    //TODO 注销播放器必须
    mnPlayControl.releasePlayer();
}
```

4.卡回放

4.1 实现播放器(查看3.1)（Implement the player)

4.2 设置播放器监听（查看3.2） (Set the player to listen)

4.3 卡录像获取与播放卡录像（Step 3: Get local card video)

```
// 获取指定时间段内的所有卡录像（Get all card videos in the specified time period）

String pszStartTime = "2020-01-18 00:00:01";
String pszEndTime = "2020-01-18 23:59:59";

MNOpenSDK.getDeviceLocalVideos(mDevice.getSn(), 0, pszStartTime, pszEndTime, new
MNOpenSDKInterface.DeviceLocalVideosCallBack() {
    @Override
    public void onDeviceLocalVideos(String videoData) {
        // videoData : 返回卡录像数据Json(Return Card Recording Data Json)

        String startTime = "2020-01-18 00:00:01";
        String endTime = "2020-01-18 23:59:59";

        // 播放指定时间段的卡录像（Play card video for a
        // specified time period）
        mnPlayControl.playTfCardVideo(videoData, startTime, endTime);
    }
});
```

4.4 播放器暂停，重新开始与销毁(查看3.4)（Pause, restart and destroy the player)

5.云回放

5.1 实现播放器(查看3.1)（Implement the player)

5.2 设置播放器监听（查看3.2） (Set the player to listen)

5.3 播放云视频（Play Cloud Video)

```
//mCloudVideo: AlarmsBean 云录像数据（Cloud recording data）
mnPlayControl.playCloudVideo(mCloudVideo);
```

5.4 播放器暂停，重新开始与销毁(查看3.4)（Pause, restart and destroy the player)

6.MNPlayControl 类说明（Method description)

6.1 MNPlayControl 方法说明（Method description)

```
 /**
 * 设置播放器初始数据(Set player initial data)
 *
 * @param deviceInfo 设备信息(Device Information)
 */
void setDeviceInfo(DevicesBean deviceInfo);

/**
  * 开启直播(Start live)
  *
  * @param stream    码流（Code stream）
  * @param channelId 通道号（Channel number）
  */
void startLive(MNControlAction.MNCodeStream stream, int channelId);

/**
 * 打开视频失败，尝试重新打开视频（Failed to open video, try to reopen video）
 */
void tryStartLive();

/**
  * 销毁播放器（Destroy player）
  */
void releasePlayer();

/**
  * 关闭播放任务（Close playback task）
  */
void stopPlayer();

/**
  * 暂停播放任务（Pause task）
  */
void pausePlayer();

/**
  * 恢复播放任务（Resume playback task）
  */
void resumePlayer();

/**
  * 开启音频（Turn on audio）
  */
void startAudio();
```

```
/**
  * 关闭音频（关闭音频）
  */
void stopAudio();

/**
  * 开始录像（Start recording）
  *
  * @param filePath 录制文件存放地址（Recording file storage address）
  * @return boool 开始录制成功或者失败
  */
boolean startRecord(String filePath);

/**
  * 结束录像(End recording)
  *
  * @return 录制成功返回文件录制文件路径（Recording successfully returns the file
recording file path）
  */
String stopRecord();

/**
  * 开启对讲(Start intercom)
  */
void startTalk();

/**
  * 结束单项对讲(End single talk)
  */
void stopTalk();

/**
  * 截图(Screenshot)
  *
  * @param fileName 文件存储位置(File storage location)
  * @return
  */
boolean screenShot(String fileName);

/**
  * 设置码流(Set the code stream)
  *
  * @param stream // 码流
  * @return
  */
boolean setCodeStream(MNControlAction.MNCodeStream stream);

/**
  * 设置值直播窗口大小（Set value live window size）
  *
  * @param width
  * @param height
  */
void setSurfaceSize(int width, int height);

/**
  * 播放云视频（Play cloud video）
  *
```

```
     * @param alarmsBean 云视频授权路径（Cloud video authorization path）
     */
    void playCloudVideo(AlarmsBean alarmsBean);

    /**
     * 配置自动播放24小时云录像(Configure automatic playback of 24-hour cloud
    recording)
     *
     * @param record24Beans   24小时报警数组（24-hour alarm array）
     * @param vStartLoaclTime 开始播放时间（Start time）
     * @param vEndLoaclTime   结束播放时间（End play time）
     */
    void config24HCloudRecordAutoTask(List<Record24Bean> record24Beans, String
    vStartLoaclTime, String vEndLoaclTime);

    /**
     * 切换播放时间段（24小时云录像）（Switch playback time period (24-hour cloud
    video)）
     *
     * @param record24Beans   24小时报警数组（24-hour alarm array）
     * @param vStartLoaclTime 开始播放时间（Start time）
     * @param vEndLoaclTime   结束播放时间（End play time）
     */
    void change24CloudRecordAutoPlayTime(List<Record24Bean> record24Beans, String
    vStartLoaclTime, String vEndLoaclTime);

    /**
     * 下载云视频(Download Cloud Video)
     *
     * @param savePath   录像保存路径(Video save path)
     * @param alarmsBean 文件名称(file name)
     */
    void downCloudAlarmVideo(String savePath, AlarmsBean alarmsBean);

    /**
     * 播放卡录像(Play card video)
     *
     * @param recordJson 卡录像数据(Card recording data)
     * @param startTime  开始播放时间(Start play time)
     * @param endTime    结束播放时间(End play time)
     */
    void playTfCardVideo(String recordJson, String startTime, String endTime);

    /**
    * 变化播放卡录像时间(Change playback card recording time)
    *
    * @param recordJson 卡录像数据(Card recording data)
    * @param startTime  开始播放时间(Start play time)
    * @param endTime    结束播放时间(End play time)
    */
    void playTfCardVideoChange(String recordJson, String startTime, String endTime);

    /**
    * 下载卡录像视频(Download card video)
    *
    * @param recordJson   卡录像数据(Card recording data)
    * @param startTime    下载开始时间(Download start time)yyyy-MM-dd HH:mm:ss
    * @param stopTime     下载结束时间(Download end time) yyyy-MM-dd HH:mm:ss
```

```
* @param downloadPath 视频保存地址 (Video save address)
*/
void downloadTfCardVideo(String recordJson, String startTime, String stopTime,
String downloadPath);

/**
* 云台控制(PTZ control)
*
* @param direction 方向
* @param channelId 通道号 0
*/
void ptzControl(MNControlAction.MNDirection direction, int channelId);

/**
* 设置设备是否支持云台控制(Set whether the device supports PTZ control)
*
* @param support 是否支持云台控制
*/
void supportPTZControl(boolean support);

/**
* 摇头机返回原点(Shaking machine returns to the origin)
*/
void ptzResetAngle();

/**
* 保存摇头机预置点(Save the shaking machine preset)
*
* @param pointName  收藏名称（Favorite name）
* @param pointIndex (Favorites index：{0,1,2,3,4,5})
*/
void saveFavoritePoint(String pointName, String pointIndex);

/**
* 前往指定收藏点（Go to designated collection point）
*
* @param pointIndex (Favorites index：{0,1,2,3,4,5})
*/
void gotoFavoritePoint(String pointIndex);

/**
* 删除摇头机预置点（Delete the preset point of the shaker）
*
* @param pointId    收藏点id （Favorites id）
* @param pointIndex (Favorites index：{0,1,2,3,4,5})
*/
void delFavoritePoint(String pointId, String pointIndex);
```

6.2 MNPlayControl 任务回调说明（Task callback description)

```
public interface MNPlayControlLinstener {
    /**
     * 视频状态回调（Video status callback）
     *
     * @param videoType  播放视频类型（Play video type）
```

```
     * @param taskStatus 任务状态（Task status）
     * @param fProgress  进度（如果是下载任务）（Progress (if it is a download
task)）
     */
    void OnTaskStatus(int videoType, int taskStatus, float fProgress);

    /**
     * @param taskStatus 任务创建是否成功（Whether the task was created
successfully）
     * @param downStatus 下载状态: 1：下载中，2：下载失败 ，3：下载成功，4：取消下载
（Download          * status: 1: downloading, 2: download failed, 3: download
successful,
     * 4: cancel download）
     * @param progress   下载进度(Download progress)
     * @param fileName   文件路径(file path)
     */
    void onDownloadTaskStatus(boolean taskStatus, int downStatus, float
progress, String fileName);

    /**
     * 摇头机收藏预置点回调(Shaking machine collection presets callback)
     *
     * @param result  设置结果(Setting results)
     * @param pointId 预置点(Preset)
     */
    void onSetSessionCtrl(boolean result, String pointId);

    /**
     * 摇头机删除收藏点回调（Shaking machine delete favorites callback）
     *
     * @param result  设置结果(Setting results)
     * @param pointId 预置点(Preset)
     */
    void onDelSessionCtrl(boolean result, String pointId);

    /**
     * 前往收藏点（Go to favorites）
     *
     * @param result  设置结果(Setting results)
     * @param pointId 预置点(Preset)
     */
    void onGotoSessionCtrl(boolean result, String pointId);

    /**
     * 视频帧率与网速回调（Video frame rate and internet speed callback）
     *
     * @param nYear                  当前播放视频中解析到的时间:年
     * @param nMonth                 当前播放视频中解析到的时间:月
     * @param nDay                   当前播放视频中解析到的时间:日
     * @param nHour                  当前播放视频中解析到的时间:时
     * @param nMinute                当前播放视频中解析到的时间:分
     * @param nSecond                当前播放视频中解析到的时间:秒
     * @param lNetworkFlowPerSecond 网速
     */
    void runSpeed(int nYear, int nMonth, int nDay, int nHour, int nMinute, int
nSecond, long lNetworkFlowPerSecond);

    /**
```

```
     * 音频开关状态变化（Audio switch status change）
     *
     * @param isOpen
     */
    void onAudioSwitchChanged(boolean isOpen); // 音频开关变化

    /**
     * 语音对讲状态变化(Voice intercom status changes)
     *
     * @param isOpen
     */
    void onHoldTalkSwitchChanged(boolean isOpen); // 单攻对讲变化

    /**
     * 摇头机转动到达边界回调(Shaking machine turns to reach the boundary callback)
     */
    void onToborder();

    /**
     * NVR 设备会有这个回调(NVR devices will have this callback)
     *
     * @param data
     */
    void onNvrSessionView(String data);

    /**
     * 输入音量大小
     *
     * @param volume
     */
    void onAudioVolume(int volume);
}
```

MNKit

1. 账号相关（Account related）

1.1 账号登录（Account login）

```
/**
* 账号登录（Account login）
*
* @param username 用户名
* @param password 密码
* @param callback Callback method
*/
public static void loginWithAccount(String username, String password,
LoginCallBack callback) ;

/**
* Callback method
*/
public interface LoginCallBack {
    void onLoginFailed(String msg);
    void onLoginSuccess(LoginBean response);
```

```
}
```

LoginBean解析：

字段名	类型	字段介绍
code	int	消息码：2000 OK 操作成功; 3001 invalid app_key or app_secret 无效的app_key和app_secret; 3002 This user is not activated 该用户没有被激活; 5000 username or password error 用户名或密码错误;
msg	String	错误信息
user_id	String	用户ID
access_token	String	服务分配的全局唯一接口调用凭据

1.2 获取注册账号验证码(Get registered account verification code)

```
/**
* 获取注册账号验证码(Get registered account verification code)
*
* @param country_code 国家码（手机号码注册有效）(Country code (mobile phone number registration is valid))
* @param phone        手机号码(mobile phone number)
* @param email        电子邮箱（E-mail）
* @param locale       语言 "en_US"或"zh_CN"（Language "en_US" or "zh_CN"）
* @param valid        注册方式  "sms"或者"email"（Registration method "sms" or "email"）
* @param callBack     Callback method
*/

public static void getAuthcode(String country_code, String phone, String email, String locale, String valid, MNKitInterface.AuthcodeCallBack callBack);

//Callback method
public interface AuthcodeCallBack {
    void onGetAuthcodeSuc(BaseBean result);
    void onGetAuthcodeFailed(String msg);
}
```

1.3用户注册（New User Registration）

```
/**
* 新用户注册（New User Registration）
*
* @param email       注册邮箱（与手机号码注册相斥）（Registered email (excludes registration with mobile number)）
* @param phone       注册手机号码（与注册邮箱相斥）（Registered mobile number (excludes registered email)）
* @param active_code 对应注册方式的验证码（Verification code corresponding to registration method）
* @param callBack    Callback method
*/
```

```
public static void regiterUser(String email, String phone, String active_code,
MNKitInterface.RegiterUserCallBack callBack);

//Callback method
public interface RegiterUserCallBack {
    void onRegiterUserSuc(BaseBean result);
    void onRegiterUserFailed(String msg);
}
```

1.4新用户设置密码（Set password for new user）

```
/**
* 为新用户设置密码（Set password for new user）
*
* @param email        邮箱用户（与手机号码注册相斥）（Email user (excludes
registration with mobile number)）
* @param phone        手机号码用户（与注册邮箱相斥）（Mobile number user (excludes
registered email)）
* @param password     用户密码 （user password）
* @param country_code 国家码（Country code）
* @param active_code  验证码（Verification code）
* @param callBack     Callback method
*/

public static void setUserPassword(String email, String phone, String password,
String country_code, String active_code, MNKitInterface.SetUserPasswordCallBack
callBack);

//Callback method
public interface SetUserPasswordCallBack {
    void onSetUserPasswordSuc(BaseBean result);
    void onSetUserPasswordFailed(String msg);
}
```

2. 设备相关（Device related）

2.1 绑定设备（Bind device）

```
/**
* 绑定设备(Bind device)
*
* @param sn       Device SN
* @param vn       Device VN(默认ABCDEF)
* @param callBack Callback method
*/
public static void bindDeviceBySnAndVn(String sn, String vn,
DeviceBindViewCallBack callBack);
/**
* Callback method
*/
public interface DeviceBindViewCallBack {
    void onBindDeviceFailed(String message);
    void onBindDeviceSuc(BaseBean response);
}
```

BaseBean：

字段名（Field name）	类型(Type)	字段介绍(Field introduction)
code	int	消息码：2000 OK 操作成功; 3000 invalid access_token,无效的访问令牌; 3001 invalid app_key or app_secret,无效的app_key和app_secret; 5000 bind device error,绑定设备失败; 5001 device has been bound by other users 设备已被其他用户绑定; 5002 device not exist,设备不存在; 5003 vn error,验证码错误;
msg	String	错误信息

2.2 绑定分享设备（Binding sharing device）

```
/**
* 绑定分享设备(Binding sharing device)
*
* @param invite_code 邀请码(Invitation code)
* @param callBack    Callback method
*/
public static void bindShareDeviceByInviteCode(String invite_code,
BindShareDeviceViewCallBack callBack);
/**
* Callback method
*/
public interface BindShareDeviceViewCallBack {
    void onBindShareDeviceViewFailed(String msg);
    void onBindShareDeviceViewSuc(BaseBean response);
}
```

2.3 解除账号和设备的绑定关系（Unbind account and device）

```
/**
* 解除账号和设备的绑定关系（Unbind account and device）
*
* @param sn       Device SN
* @param callBack Callback method
*/
public static void unbindDevice(String sn, UnbindDeviceCallBack callBack);
/**
* Callback method
*/
public interface UnbindDeviceCallBack {
    void onUnbindDeviceFailed(String message);
    void onUnbindDeviceSuc(BaseBean response);
}
```

2.4 主账号取消分享设备（Active account cancellation）

```
/**
* 主账号主动取消分享设备（Active account cancellation）
```

```
*
* @param device_id Device ID
* @param user_id   user id
* @param account   account
* @param callBack  Callback method
*/
public static void cancelSharedDevice(String device_id, String user_id, String
account, CancelShareDeviceCallBack callBack);

/**
* Callback method
*/
public interface CancelShareDeviceCallBack {
    void onCancelShareDeviceFailed(String localizedMessage);
    void onCancelShareDeviceSuc(BaseBean response);
}
```

2.5 被分享用户解除分享设备（Unshared device by shared user）

```
/**
* 被分享用户解除分享设备（Unshared device by shared user）
*
* @param device_id Device ID
* @param callBack  Callback method
*/
public static void unbindSharedDevice(String device_id,
UnBindShareDeviceCallBack callBack);

/**
* Callback method
*/
public interface UnBindShareDeviceCallBack {
    void onUnBindShareDeviceFailed(String localizedMessage);
    void onUnBindShareDeviceSuc(BaseBean response);
}
```

2.6 主账号设置分享设备的权限（Master account reset permissions for sharing devices）

```
/**
* 主账号设置分享设备的权限（Master account reset permissions for sharing devices）
*
* @param device_id Device ID
* @param user_id   被分享账号ID（Shared account ID）
* @param authority 新的权限（New permissions）
* @param callback  Callback method
*/
public static void updateShareDeviceAuthority(String device_id, String user_id,
int authority, UpdateShareDeviceAuthorityCallBack callback);

/**
* Callback method
*/
public interface UpdateShareDeviceAuthorityCallBack {
```

```
    void onUpdateShareDeviceAuthoritySuc();
    void onUpdateShareDeviceAuthorityFailed(String msg);
}
```

2.7 修改设备名称(Modify device name)

```
/**
* 修改设备名称(Modify device name)
*
* @param sn       Device SN
* @param devName  New device name
* @param callBack Callback method
*/
public static void modifyDeviceNameWithSN(String sn, String devName,
ModifyDeviceNameCallBack callBack);


/**
* Callback method
*/
public interface ModifyDeviceNameCallBack {
    void onModifyDeviceNameFailed(String message);
    void onModifyDeviceNameSuc(BaseBean response);
}
```

2.8 上传设备封面(Upload device cover)

```
/**
* 上传设备封面(Upload device cover)
*
* @param file      File
* @param sn        Device SN
* @param channelId channel Id
* @param callBack  Callback method
*/
public static void updateDeviceCover(File file, String sn, int channelId,
UpdateDeviceCoverCallBack callBack) ;

/**
* Callback method
*/
public interface UpdateDeviceCoverCallBack {
    void onUpdateDeviceCoverFailed(String msg, int id);
    void onUpdateDeviceCoverSuc(UpdateDeviceCoverBean response, int id);
}
```

UpdateDeviceCoverBean:

字段名（Field name)	类型(Type)	字段介绍(Field introduction)
code	int	消息码：2000 OK 操作成功; 3001 invalid app_key or app_secret,无效的app_key和app_secret;
msg	String	错误信息
url	String	封页地址

2.9 获取用户设备信息(Get user device information)

```
/**
* 获取设备列表(Get device list)
*
* @param callBack Callback method
*/
public static void getDevicesList(DevListCallBack callBack);
//Callback method
public interface DevListCallBack {
    void onGetDevListFailed(String msg);
    void onGetDevListSuccess(DevListSortBean msg);
}

/**
* 获取单个设备详情(Get individual device details)
* @param sn 设备SN
* @param callBack
*/
public static void getDeviceBySn(String sn, GetDevceBySnCallBack callBack);
//Callback method
public interface GetDevceBySnCallBack {
    void onGetDevceBySnFailed(String msg);
    void onGetDevceBySnSuc(DeviceInfoBean response);
}
```

DevListSortBean:

字段名 (Field name)	类型(Type)	字段介绍(Field introduction)
code	int	消息码：2000 OK 操作成功; 3001 invalid app_key or app_secret,无效的app_key和app_secret;
msg	String	错误信息
devices	ArrayList<DevicesBean>	设备列表信息

DevicesBean device 列表内字段说明（Field descriptions in the list）：

字段名（Field name)	类型(Type)	字段介绍(Field introduction)
id	String	设备唯一识别ID,服务属性(Device unique identification ID, service attribute)
sn	String	设备唯一识别ID,设备属性(Device unique identification ID, device attributes)
dev_name	String	设备名称(Device name)
online	int	设备是否在线：0表示不在线， 1表示在线 2 表示休眠状态(Whether the device is online: 0 means offline, 1 means online, 2 means hibernation)
logo	String	设备封面logo(Device cover logo)
firm_id	String	设备所属厂商(Device manufacturer)
ver	String	设备当前固件版本号(Device current firmware version number)
model	String	设备型号(Device model)
storage_received	int	云存储领取状态(0 未领取任何云存服务 1 已经领取过免费云存服务)
first_online_time	String	设备首次在线时间(Device first online time)
last_offline_time	String	设备最后离线时间(Device last offline time)
support_ability	SupportAbilityBean(Object)	设备设备所具备的能力集(Capabilities set of Device)

SupportAbilityBean (support_ability字段说明)（Field description）：

字段名（Field name)	类型(Type)	字段介绍(Field introduction)
h24recordAbility	int	1:支持24小时云录像(Support 24-hour cloud recording);0:不支持（not support)
timingCaptureAbility	int	1:支持定时上报图片(Support to report pictures regularly);0:不支持(not support)
batteryAbility	int	1:是电池供电设备(Is battery powered);0:非电池供电(Non-battery)
ptzAbility	PtzAbilityBean(Object)	direction=1:支持云台控制(Support PTZ control);0:不支持云台控制(Does not support PTZ control)
fourgAbility	FourgAbilityBean(Object)	fourgEnable=1:支持4G流量卡设备(Support 4G traffic card device);0:不支持4G流量卡(Support 4G traffic card)
cloudStorageAbility	CloudStorageAbilityBean(Object)	eventStorage=1:支持事件云存储(Support event cloud storage);eventStorage=0:不支持事件云存储(Event cloud storage is not supported)。h24recordStorage=1：支持24小时云存储(Support 24-hour cloud storage);h24recordStorage=0:不支持24小时云存储(Does not support 24-hour cloud storage)
alarmAbility	List(Object)	设备支持的报警消息类型(Alarm message types supported by the device)

2.10 获取账号下已分享给他人观看的设备

```
/**
* 获取账号下已分享给他人观看的设备（Get devices that have been shared with others for viewing under your account）
*
* @param callback Callback method
*/
public static void getShareToOtherDevLists(GetShareDevListsCallBack callback);

//Callback method
public interface GetShareDevListsCallBack {
    void onGetShareDevListsSuc(MeToOtherBean bean);
    void onGetShareDevListsFailed(String msg);
}
```

MeToOtherBean 说明（Description）：

字段名（Field name)	类型(Type)	字段介绍(Field introduction)
code	消息码：2000 OK 操作成功; 3001 invalid app_key or app_secret,无效的app_key和app_secret;	

| msg| String |错误信息
| devices | ArrayList<DevicesBean> | 设备列表信息同2.9(Device list information is the same as 2.9) |

2.11 获取他人分享给我的设备

```
/**
* 获取他人分享给我的设备（Get devices that others have shared with me）
*
* @param callback Callback method
*/
public static void getShareOtherToMeDevLists(GetOhterShareDevListsCallBack
callback);

// Callback method
public interface GetOhterShareDevListsCallBack {
    void onGetOhterShareDevListsSuc(OtherToMeBean response);
    void onGetOhterShareDevListsFailed(String msg);
}
```

OtherToMeBean 说明（Description）：

字段名（Field name)	类型(Type)	字段介绍(Field introduction)
code	消息码：2000 OK 操作成功; 3001 invalid app_key or app_secret,无效的app_key和app_secret;	

| msg| String |错误信息
| devices | ArrayList<DevicesBean> | 设备列表信息同2.9(Device list information is the same as 2.9) |

2.12 获取对应设备绑定信息与在线状态

```
/**
* 获取对应设备绑定信息与在线状态(et the corresponding device binding information and
online status)
*
* @param sn       Device SN
* @param callback Callback method
*/
public static void getDeviceStateInfoWithSn(String sn, DeviceStateInfoCallBack
callback);

// Callback method
```

```
public interface DeviceStateInfoCallBack {
    void onGetDeviceStateFailed(String message);
    void onGetDeviceStateSuc(DevStateInfoBean response);
}
```

DevStateInfoBean 说明（Description）：

字段名（Field name)	类型(Type)	字段介绍(Field introduction)
code	消息码：2000 OK 操作成功;3001 invalid app_key or app_secret,无效的app_key和app_secret;	
msg	String	错误信息
type	int	设备类型 参见设备详情2.9(Device type See device details, same as 2.9)
online	int	设备在线状态 0:不在线 1:在线 2:休眠(Device online status 0: Not online 1: Online 2: Sleep)
bind_state	int	设备绑定状态 0:没有被绑定; 1:被自己绑定; 2:被别人绑定;(Device binding status 0: not bound; 1: bound by itself; 2: bound by others;)
bind_user	BindUserBean(Object)	用户信息:phone[^String] , email[^String]

```
/**
* 获取设备在线状态(Get device online status)
* @param sn       Device SN
* @param callBack Callback method
*/
public static void getDeviceOnlineInfoWithSn(String sn, DevOnlineStateCallBack
callBack);

// Callback method
public interface DevOnlineStateCallBack {
    void onGetOnLineStateSucc(DevOnlineBean result);
    void onGetOnLineStateFailed(String msg);
}
```

DevOnlineBean 说明（Description）：

字段名（Field name)	类型(Type)	字段介绍(Field introduction)
code	消息码：2000 OK 操作成功; 3001 invalid app_key or app_secret,无效的app_key和app_secret;	
msg	String	错误信息
state	int	设备在线状态 0:不在线 1:在线 2:休眠 (Device online status 0: Not online 1: Online 2: Sleep)

2.13 获取已经接受设备分享的用户基础信息(Get basic user information that has been shared by the device)

```
/**
* 获取已经接受设备分享的用户基础信息(Get basic user information that has been shared by the device)
*
* @param device_id Device Id
*/
public static void getInviteShareUsers(String device_id, MNKitInterface.GetInviteShareUsersCallBack callback);

// Callback method
public interface GetInviteShareUsersCallBack {
    void onGetInviteShareUsersSuc(ShareUserListBean response);
    void onGetInviteShareUsersFailed(String msg);
}
```

ShareUserListBean 说明（Description）：

字段名（Field name)	类型(Type)	字段介绍(Field introduction)
code	消息码：2000 OK 操作成功; 3001 invalid app_key or app_secret,无效的app_key和app_secret;	
msg	String	错误信息
limit	int	设备可分享的最大用户数量 (Maximum number of users the device can share)
users	Lis(ShareUserBean)	分享用户信息(Share user information)

ShareUserBean 说明（Description）：

字段名（Field name)	类型(Type)	字段介绍(Field introduction)
id	String	分享用户分配ID(Share user assigned ID)
device_id	String	分享设备id(Share device id)
username	String	分享用户名(hare username)
email	String	分享用户为邮箱的用户名(Username for sharing user as mailbox)
phone	String	分享用户为手机的用户名(Share user as mobile phone username)
avatar	String	分享用户头像(Share user avatar)
nickname	String	分享用户昵称(Share user nickname)
realname	String	
state	String	设备分享状态0:等待接受分享；1:已经接受分享(Device sharing status 0: waiting to accept sharing; 1: sharing accepted)
watch_time	String	分享设备时间(Share device time)
authority	String	分享设备权限(Share device permissions)
remain_time	String	分享设备到期时间(Share device expiration time)

2.14 获取某个设备的分享历史（Get the sharing history of a device)

```
/**
* 获取某个设备的分享历史（Get the sharing history of a device）
*
* @param sn       Device SN
* @param callback Callback method
*/
public static void getSharedHistoryBySn(String sn,
MNKitInterface.GetSharedHistoryCallBack callback);

//Callback method
public interface GetSharedHistoryCallBack {
    void onGetSharedHistoryFailed(String msg);
    void onGetSharedHistorySuc(SharedHistoryBean response);
}
```

SharedHistoryBean 说明（Description）：

字段名（Field name)	类型(Type)	字段介绍(Field introduction)
code	消息码：2000 OK 操作成功; 3001 invalid app_key or app_secret,无效的app_key和app_secret;	
msg	String	错误信息
list	List(ListBean)	分享用户信息(Share user information)

SharedHistoryBean.ListBean 说明（Description）：

字段名（Field name)	类型(Type)	字段介绍(Field introduction)
user_name	String	分享用户名(Share user username)
share_time	String	分享设备时间(Share device time)
start_time	String	开始分享设备的时间(Time to start sharing the device)
end_time	String	分享设备的到期时间(Share device expiration time)

2.15 分享设备到指定账号（Share device to specified account)

```
/**
* 分享设备到指定账号（Share device to specified account）
*
* @param sn           Device SN
* @param time_limit   分享有效时间（Share effective time）
* @param authority    被分享人可以查看的权限（Shared permissions that people can view）
* @param account      被分享人账号（手机账号/邮箱账号）（Shared account (mobile phone account)）
* @param countryCode 国家码(Country Code)
* @param locale       语言（language "zh_CN" 或 "en_US"）
* @param callback     Callback method
*/
public static void shareDevToAccount(String sn, int time_limit, int authority,
String account, String countryCode, String locale,
MNKitInterface.ShareDevToAccountCallBack callback);

//Callback method
public interface ShareDevToAccountCallBack {
    void onSharedDevToAccountFailed(String msg);
    void onSharedDevToAccountSuc(BaseBean baseBean);
}
```

2.16 获取分享设备的二维码(Get the QR code of the sharing device)

```
/**
* 获取分享设备的二维码(Get the QR code of the sharing device)
*
* @param device_id  Device Id
* @param authority  被分享人可以查看的权限（Shared permissions that people can
view）
* @param time_limit 分享有效时间（Share effective time）
* @param callback   Callback method
*/
public static void getShareDevQrCode(String device_id, int authority, int
time_limit, MNKitInterface.GetShareDevQrCodeCallBack callback);

// Callback method
public interface GetShareDevQrCodeCallBack {
    void onGetShareDevQrCodeSuc(byte[] bytes);
    void onGetShareDevQrCodeFailed(String msg);
}
// E.g
public void onGetShareDevQrCodeSuc(byte[] bytes){
    Bitmap bmp = BitmapFactory.decodeByteArray(bytes, 0, bytes.length);
}
```

2.17 获取等待接受分享的设备列表(Get list of devices waiting to be shared)

```
/**
* 获取等待分享的设备列表
*
* @param callback
*/
public static void getWaitingSharedDev(GetShareWaitingDevCallBack callback)

// Callback method
public interface GetShareWaitingDevCallBack {
    void onGetShareWaitingDevFailed(String msg);
    void onGetShareWaitingDevSuc(WaitingShareDevBean response);
}
```

WaitingShareDevBean 说明（Description）：

字段名（Field name)	类型(Type)	字段介绍(Field introduction)
code	消息码：2000 OK 操作成功; 3001 invalid app_key or app_secret,无效的app_key和app_secret;	
msg	String	错误信息
share_devices	List(ShareDevicesBean)	分享设备信息(Share device information)

ShareDevicesBean 说明（Description）：

字段名（Field name）	类型(Type)	字段介绍(Field introduction)
sn	String	分享设备SN(Share device SN)
device_user	DeviceUserBean	分享用户基础信息(Share user basic information)

DeviceUserBean 说明（Description）：

字段名（Field name）	类型(Type)	字段介绍(Field introduction)
phone	String	分享用户为手机用户的电话号码(Share the phone number of the mobile user)
email	String	分享用户为邮箱用户的邮箱地址(Email address of the sharing user as a mailbox user)

2.18 接受或拒绝使用分享设备(Accepting or rejecting shared devices)

```
/**
* 接受或拒绝使用分享设备
* @param sn 分享设备的SN（Share device's SN）
* @param received 1：接受使用分享设备；0：拒绝使用分享设备（1: Accept sharing device;
0: Decline sharing device）
* @param callback
*/
public static void receivedSharingDevice(String sn, int received,
ReceivedShareDeviceCallBack callback)

// Callback method
public interface ReceivedShareDeviceCallBack {
    void onReceivedShareDeviceSuc(BaseBean response, int received);
    void onReceivedShareDeviceFailed(String msg, int received);
}
```

3. 云视频

报警消息类型定义（Alarm message type definition）

```
public class MnKitAlarmType {

    /**
     * 全部报警（All alarm messages）
     */
    public static int AllAlarm_Detection = 1 << 0;

    /**
     * 移动侦测报警（Motion detection alarm）
     */
    public static int Motion_Detection = 1 << 1;
```

```
    /**
     * 人形检测报警（Human detection alarm）
     */
    public static int Humanoid_Detection = 1 << 2;
    /**
     * 视频遮挡报警（Video blocking alarm）
     */
    public static int Occlusion_Detection = 1 << 3;
    /**
     * 人脸识别报警（Face recognition alarm）
     */
    public static int Face_Detection = 1 << 4;

    /**
     * 哭声检测报警（Crying detection alarm）
     */
    public static int Cry_Detection = 1 << 5;
    /**
     * 箱体报警（Box alarm）
     */
    public static int Box_Detection = 1 << 6;
    /**
     * 门铃来电提醒报警（Doorbell call alert）
     */
    public static int CallReminder_Detection = 1 << 7;
    /**
     * PIR 报警（Human infrared detection alarm）
     */
    public static int Infrared_detection = 1 << 8;
    /**
     * 外部IO报警（External IO alarm）
     */
    public static int IO_detection = 1 << 9;

}
```

3.1 获取云存报警信息

```
/**
* 获取云存报警信息(Get cloud storage alarm information)
*
* @param deviceSns  // 设备SN列表(Device SN list)
* @param startTime  // 查询开始时间(Query start time)
* @param endTime    // 查询结束时间(Query end time)
* @param alarmTypeOptions // 查询报警信息类型(Query alarm information type)
* @param personName // 名字，人脸识别有效(Name, face recognition is valid)
* @param pageStart  // 第几页，从0开始，开始时间与结束时间相同的查询有效(Page number,
starting from 0, the query with the same start time and end time is valid)
* @param pageSize   // 每页查询条数(Number of queries per page)
* @param callBack   Callback method
*/
public static void getCloudAlarmData(ArrayList<String> deviceSns, long
startTime, long endTime, int alarmTypeOptions,
String personName, int pageStart, int pageSize, CloudAlarmsCallBack callBack) ;

// Callback method
public interface CloudAlarmsCallBack {
```

```
    void onGetCloudAlarmsFailed(String msg);
    void onGetCloudAlarmsSuc(CloudAlarmsBean response);
}
// Example

// 设置报警类型,同时获取 移动侦测，人形报警，遮挡报警与人脸识别报警信息
int alarmTypeoptions = MnKitAlarmType.Motion_Detection |
MnKitAlarmType.Humanoid_Detection | MnKitAlarmType.Occlusion_Detection |
MnKitAlarmType.Face_Detection;

MNKit.getCloudAlarmData(deviceSns, startSearchTime, endSearchTime,
alarmTypeoptions, null, 0, 20, this);
```

CloudAlarmsBean 说明（Description）：

字段名（Field name)	**类型(Type)**	**字段介绍(Field introduction)**
code	消息码：2000 OK 操作成功; 3001 invalid app_key or app_secret,无效的app_key和app_secret;	
msg	String	错误信息
alarms	ArrayList(AlarmsBean)	云报警信息详情(Cloud alarm message details)

AlarmsBean 说明（Description）：

字段名（Field name）	类型 (Type)	字段介绍(Field introduction)
alarmId	String	云报警消息id
deviceSn	String	产生云报警消息的设备SN
alarmTime	long	产生云报警消息的时间
alarmType	int	报警消息主类型
subAlarmType	int	报警消息子类型
imageUrl	String	报警图片
videoUrl	String	普通事件报警视频与recordUrl互斥
vStartTime	long	普通事件报警视频开始时间
vEndTime	long	普通事件报警视频结束时间
recordUrl	String	24小时录像的时间报警与videoUrl互斥
vStartLocalTime [^yyyy-MM--dd HH:mm:ss]	String	24小时录像报警视频开始时间
vEndLocalTime[^yyyy-MM--dd HH:mm:ss]	String	24小时录像报警视频结束时间
devImagePath	String	4G设备本地卡存储地址,与videoUrl和recordUrl都互斥

3.2 普通云存报警信息URL信息鉴权

```
/**
* 对报警视频URL信息鉴权(Authentication of alarm video URL information)
*
* @param url  需要鉴权的视频地址(Video address requiring authentication)
* @param callBack Callback method
*/
public static void authenticationUrl(String url,
MNKitInterface.AuthenticationUrlCallBack callBack);

// Callback method
public interface AuthenticationUrlCallBack {
    void onAuthenticationUrlSuc(AuthenticationBean bean);
    void onAuthenticationUrlFailed(String msg);
}
```

AuthenticationBean 说明（Description）：

字段名（Field name）	类型(Type)	字段介绍(Field introduction)
code	int	消息码：2000 OK 操作成功; 3001 invalid app_key or app_secret,无效的app_key和app_secret;
msg	String	错误信息
urls	ArrayList(UrlsBean)	云报警信息鉴权结果(Cloud alarm information authentication result)

UrlsBean 说明（Description）：

字段名（Field name）	类型(Type)	字段介绍(Field introduction)
expiration	int	
file_size	int	文件大小(File size)

3.3 获取24小时云录像报警信息

```
/**
* 获取24小时云录像报警信息(Get 24-hour cloud recording alarm information)
*
* @param sn          Decice SN
* @param start_time 查询开始时间(Query start time)yyyy-MM-dd HH:mm:ss
* @param end_time    查询结束时间(Query end time)yyyy-MM-dd HH:mm:ss
* @param callback    Callback method
*/
public static void get24HCloudRecord(String sn, String start_time, String
end_time, Get24HCloudRecordCallBack callback);

// Callback method
public interface Get24HCloudRecordCallBack {
    void onGet24HCloudRecordFailed(String msg);

    void onGet24HCloudRecordSuc(Record24AlarmBean response);
}
```

Record24AlarmBean 说明（Description）：

字段名（Field name）	类型(Type)	字段介绍(Field introduction)
id	String	云报警消息id
deviceId	String	产生云报警消息的设备id
channelNo	int	产生报警消息的设备通道号
videoUrl	String	24小时录像的报警视频地址
startTime [^yyyy-MM--dd HH:mm:ss]	String	24小时录像报警视频开始时间
endTime[^yyyy-MM--dd HH:mm:ss]	String	24小时录像报警视频结束时间

3.4 删除报警消息(Delete alarm message)

```
/**
* 删除报警消息
*
* @param alarmIds 报警消息Id数组
*/
public static void delAlarms(ArrayList<String> alarmIds, DelAlarmsCallBack
callback)


// Callback method
public interface DelAlarmsCallBack {
    void onDelAlarmsSuc();
    void onDelAlarmsFailed(String msg);
}
```

3.5 删除对应时间段的报警消息(Delete alarm messages for the corresponding time period)

```
/**
* 删除对应时间段的报警消息
*
* @param deviceSns  设备SN数组
* @param startTime  删除报警消息的起始时间
* @param endTime    删除报警消息的结束时间
* @param alarmTypeoptions  要删除报警消息的类型 int alarmTypeoptions =
*       MnKitAlarmType.Motion_Detection | MnKitAlarmType.Face_Detection;
* @param personName 人脸识别消息类型中对应名字的人
* @param callback   回调方法
*/
public static void delAlarmsByTime(ArrayList<String> deviceSns, long startTime,
long endTime, int alarmTypeoptions, String personName, DelAlarmsByTimeCallBack
callback)

// Callback method
public interface DelAlarmsByTimeCallBack {
    void onDelAlarmsByTimeSuc();
    void onDelAlarmsByTimeFailed(String msg);
}
```

AlarmTypeBean 说明（Description）：

字段名（Field name）	类型(Type)	字段介绍(Field introduction)
alarmType	int	报警消息主类型
subAlarmType	int	报警消息子类型

3.6 标记报警消息状态(Flag alarm message status)

```
 /**
 * 标记报警消息状态
 *
 * @param alarmIds 报警消息Id数组
 * @param status   标记值 0：未读，1：已读
 */
 public static void modifyStates(ArrayList<String> alarmIds, int status,
AlarmModifyStateCallBack callback)

 // Callback method
public interface AlarmModifyStateCallBack {
    void onModifyStateFailed(String string);
    void onModifyStateSuc();
}
```

3.7 标记对应时间段的报警消息的已读状态(Mark the read status of alarm messages for the corresponding time period)

```
 /**
 * 标记对应时间段的报警消息的已读状态
 *
 * @param deviceSns  设备SN数组
 * @param startTime  标记报警消息的起始时间
 * @param endTime    标记报警消息的结束时间
 * @param alarmTypeoptions  要标记报警消息的类型 int alarmTypeoptions =
*       MnKitAlarmType.Motion_Detection | MnKitAlarmType.Face_Detection;
 * @param personName 人脸识别消息类型中对应名字的人
 * @param callback   回调方法
 */
public static void modifyStatesByTime(ArrayList<String> deviceSns, long
startTime, long endTime, int alarmTypeoptions, String personName, int status,
AlarmModifyStateByTimeCallBack callback)

 // Callback method
public interface AlarmModifyStateByTimeCallBack {
    void onModifyStateByTimeFailed(String string);
    void onModifyStateByTimeSuc();
}
```

4. 消息推送（Message push）

4.1 获取设备消息推送配置(Get device message push configuration)

```
/**
* 获取设备消息推送配置(Get device message push configuration)
*
* @param sn        Device SN
* @param channelId channel Id
* @param callBack  Callback method
*/
public static void getPushConfigWithSN(String sn, int channelId,
MNKitInterface.GetDevPushconfigCallBack callBack);
```

```
// Callback method
public interface GetDevPushconfigCallBack {
    void onGetDevPushconfigFailed(String msg);
    void onGetDevPushconfigSuc(PushconfigBean response);
}
```

PushconfigBean 说明（Description）：

字段名（Field name）	类型(Type)	字段介绍(Field introduction)
code	int	消息码：2000 OK 操作成功; 3001 invalid app_key or app_secret,无效的app_key和app_secret;
msg	String	错误信息
pushenable	int	是否开启消息推送1：开启推送；0：关闭消息推送(Whether to enable message push 1: enable push; 0: disable message push)
sleepenable	int	是否开启消息推送休眠1：开启休眠；0：关闭休眠(Whether to enable message push hibernation 1: enable hibernation; 0: disable hibernation)
level	int	消息推送等级(Message push level)
stranger_push	int	
alarmTypeOptions	int	允许消息推送的消息类型 MnKitAlarmType(Message types that allow message push)
sleep_time_range	List(SleepTimeRangeBean)	推送休眠时间(sleep time)

4.2 设置设备消息推送配置(Set device message push configuration)

```
/**
* 设置设备消息推送配置(Set device message push configuration)
*
* @param sn                  Device SN
* @param channelId           channel Id
* @param alarmTypeOptions    推送消息类型配置MnKitAlarmType(Push message type
configuration)
* @param sleepenable         是否启用休眠(Whether to enable hibernation)
* @param sleepTimeRangeBean 休眠时间段(Sleep period)
* @param callBack            Callback method
*/
public static void setPushConfigerationWithSN(String sn, int channelId, int
alarmTypeOptions, int sleepenable,
ArrayList<DevPushConfigBean.PushconfigBean.SleepTimeRangeBean>
sleepTimeRangeBean, MNKitInterface.SetDevPushconfigCallBack callBack);
```

5 摇头机收藏点

5.1获取收藏点信息（Get favorite point information）

```
/**
* 获取收藏点信息（Get favorite point information）
*
* @param deviceId 设备ID（device Id）
*/
public static void getFavoritePointsInfo(String deviceId,
MNKitInterface.GetFavoritePointsInfoCallBack callback);

//Callback method
public interface GetFavoritePointsInfoCallBack {
    void onGetFavoritePointsInfoSuc(FavoritesInfoBean bean);
    void onGetFavoritePointsInfoFailed(String msg);
}
```

FavoritesInfoBean 说明（Description）：

字段名（Field name)	类型(Type)	字段介绍(Field introduction)
code	int	消息码：2000 OK 操作成功; 3001 invalid app_key or app_secret,无效的app_key和app_secret;
msg	String	错误信息
list	List(FavoritePointBean)	消息推送配置(Message push configuration)

FavoritePointBean 说明（Description）：

字段名（Field name）	类型(Type)	字段介绍(Field introduction)
id	String	收藏点id(Collection point id)
name	String	收藏点名称(Favorite point name)
position_id	int	收藏点位置索引id(Favorite point location index id)
device_id	String	设备id(Device id)
image_url	String	收藏点图片地址(Collection point picture address)

MNOpenSDK

1 基础配置

1.1 获取SDK版本信息 （get the SDK Version）

```
/**
 * 获取SDK版本信息 （get the SDK Version）
 *
 * @return 返回版本信息（Version infomation）
 */
public static String getSDKVersion() {
    return MNJni.GetVersion();
}
```

1.2 初始化SDK（Initialize the SDK）

```
/**
 * 初始化SDK（Initialize the SDK）
 *
 * @param context   Context
 * @param appKey    应用分配的appKey
 * @param appSecret 应用分配的appSecret
 */
public static void initWithKeyAndSecret(Context context, String appKey, String
appSecret);
```

1.3 获取AppKey （Get AppSecret）

```
/**
 * 获取AppKey （Get AppSecret）
 *
 * @return AppSecret
 */
public static String getAppKey();
```

1.4 获取AppSecret（ Get AppSecret）

```
/**
 * 获取AppSecret（ Get AppSecret）
 *
 * @return AppSecret
 */
public static String getAppSecret() ;
```

1.5 设置域名（Modify the Domain for test)

```
/**
 * 设置域名（Modify the Domain for test）
 */
public static void setMNKitDomain(Context context, String domain);
```

1.6 设置是否提前建立与设备的链接关系（Set whether to establish a link relationship with the device in advance)

```
/**
 * 设置是否提前建立P2p连接关系，设备会自动切换到APP所在域名（Set whether to establish the connection between the APP and the device in advance, the device will automatically switch to the domain name where the APP is located）
 *
 * @param state 是否提前建立与设备的连接关系(Whether to establish a connection relationship with the device in advance)
 */
public static void setP2pPreLinkState(Context context, boolean state) ;
```

1.7 获取当前是否提前链接P2p的状态（Gets whether the current status of P2p is linked in advance)

```
/**
 * 获取当前是否提前链接P2p的状态;
 *
 * @return boolean
 */
public static boolean getP2pPreLinkState() ;
```

1.8 获取域名（Get domain name)

```
/**
 * 获取域名（Get domain name）
 *
 * @return 域名（https://restcn.bullyun.com）
 */
public static String getDomain() ;
```

1.9 AccessToken

```
/**
* 设置AccessToken（Set AccessToken）
*
* @param token
*/
public static void setAccessToken(String token);

/**
 * 获取AccessToken（Get AccessToken）
 *
 * @return accessToken
 */
public static String getAccessToken();
```

1.10 注销SDK （Cancellation the SDK)

```
/**
 * 注销SDK （Cancellation the SDK）
 */
public static void uninitSDK() ;
```

1.11 退出SDK（Logout SDK)

```
/**
 * 退出SDK（Logout SDK）
 */
public static void logout();
```

1.12 链接设备（Link device)

```
/**
 * 与设备建立连接，建立连接成功之后才能打开视频，所以提前建立连接，可以加快打开视频的速度
 * (Establish a connection with the device, and then open the video after the
connection is established, so establishing a connection in advance can speed up
the opening of the video.)
 *
 * @param sn         Device sn
 * @param isShareDev 是否是分享设备，true:分享设备不会同步切换设备所在域名。false：自己账
号下的设备，会同步切换设备域名(Whether it is a sharing device, true: the sharing
device will not switch the domain name of the device. false: Devices under your
account will switch device domain names simultaneously)
 * @return
 */
public static void linkToDevice(String sn, boolean isShareDev);
```

1.13 登录ETS与IDM服务

```
/**
* 登录ETS与IDM服务，请确保用户已经登录之后调用。
* 如果已经使用MNKit.loginWithAccount()方法登录的用户不要在调用此方法
* (To log in to ETS and IDM services, please make sure that the user * has
called after logging in.If the user who has logged in using the *
MNKit.loginWithAccount() method does not call this method)
* @param userId
* @param accessToken
*/
public static void loginEtsAndIdm(String userId, String accessToken);
```

2 设备配置

2.1 获取设备本地卡录像信息(Get device local card recording information)

```
/**
 * 获取设备本地卡录像信息(Get device local card recording information)
 *
 * @param sn           Device sn
 * @param channelId    设备通道号(Device channel number)
 * @param pszStartTime 查询起始时间（Query start time） 2019-12-11 00:00:00
 * @param pszEndTime   查询结束时间 （Query end time）2019-12-11 23:59:59
 * @param callBack     Callback method
 */
public static void getDeviceLocalVideos(String sn, int channelId, String
pszStartTime, String pszEndTime, MNOpenSDKInterface.DeviceLocalVideosCallBack
callBack);
```

2.1 获取设备基本信息(Get basic device information)

```
/**
 * 获取设备基本信息（新协议多了TF卡名称数组和数量）(Obtain basic device information (the
new protocol adds an array and number of TF card names))
 *
 * @param sn       Device sn
 * @param callback Callback method
 */
public static void getDeviceBaseInfo(String sn,
MNOpenSDKInterface.GetDeviceBaseInfoCallBack callback);
```

2.2 获取电池设备的电量(Get battery power)

```
/**
 * 获取电池设备的电量(Get battery power)
 *
 * @param sn       Device sn
 * @param callback Callback method
 */
public static void getPowerState(String sn,
MNOpenSDKInterface.GetPowerStateCallBack callback);
```

2.3 获取设备语言信息(Get device language information)

```
/**
 * 获取设备语言信息(Get device language information)
 *
 * @param sn       Device sn
 * @param callback Callback method
 */
public static void getLanguageConfig(String sn,
MNOpenSDKInterface.LanguageConfigCallBack callback) ;
```

2.4 为设备设置语言(Get device language information)

```
/**
 * 为设备设置语言（当设备支持语言切换时有效）(Set the language for the device
(effective when the device supports language switching))
 *
 * @param sn       Device sn
 * @param language 语言(language)
 * @param callback Callback method
 */
public static void setLanguageConfig(String sn, String language,
MNOpenSDKInterface.SetLanguageConfigCallBack callback);
```

2.5 获取设备视频制式(Get device video format)

```
/**
 * 获取设备视频制式(Get device video format)
 *
 * @param sn       Device sn
 * @param callback Callback method
 */
public static void getVideoStandard(String sn,
MNOpenSDKInterface.VideoStandardConfigCallBack callback);
```

2.6 获取设备音量配置(Get device volume configuration)

```
/**
 * 获取设备音量配置(Get device volume configuration)
 *
 * @param sn       Device sn
 * @param callback Callback method
 */
public static void getAudioOutputVolume(String sn,
MNOpenSDKInterface.GetAudioOutputCallBack callback);
```

2.7 设置设备音量配置(Set device volume configuration)

```
/**
 * 设置设备音量配置(Set device volume configuration)
 *
 * @param sn       Device sn
 * @param aovBean  AudioOutputSetBean channel:通道号，单通道设备默认写0，多通道设备默认从0开始(Channel number, single-channel device writes 0 by default, multi-channel devices start from 0 by default)；AudioOutputVolume 音频输出值(Audio output value)（0--100）
 * @param callback Callback method
 */
public static void setAudioOutputVolume(String sn, AudioOutputSetBean aovBean, MNOpenSDKInterface.SetAudioOutputCallBack callback) ;
```

2.8 获取NVR设备音量配置(Get NVR device volume configuration)

```
/**
 * 获取NVR设备音量配置(Get NVR device volume configuration)
 *
 * @param sn       Device sn
 * @param channels 通道号
 * @param callback Callback method
 */
public static void getNvrAudioOutputVolume(String sn, int[] channels, MNOpenSDKInterface.GetNvrAudioOutputCallBack callback);
```

2.9 设置NVR设备音量配置(Set NVR device volume configuration)

```
/**
 * 设置NVR设备音量配置(Set NVR device volume configuration)
 *
 * @param sn       Device sn
 * @param aovBeans 音量配置信息(Volume configuration information)
 * @param callback Callback method
 */
public static void setNvrAudioOutputVolume(String sn, ArrayList<AudioOutputSetBean> aovBeans, MNOpenSDKInterface.SetNvrAudioOutputCallBack callback) ;
```

2.10 获取设备静音和离线语音配置(Get device mute and offline voice configuration)

```
/**
 * 获取设备静音和离线语音配置(Get device mute and offline voice configuration)
 *
 * @param sn       Device sn
 * @param callback Callback method
 */
public static void getSoundModeConfig(String sn, MNOpenSDKInterface.GetSoundModeConfigCallBack callback);
```

2.11 设置设备静音和离线语音配置（Set device mute and offline voice configuration)

```
/**
 * 设置设备静音和离线语音配置（Set device mute and offline voice configuration）
 *
 * @param sn          Device sn
 * @param silentMode  是否开启语言提示(Whether to enable the language prompt)
 * @param voiceEnable 是否关闭离线语音提示(hether to turn off offline voice
prompts)
 * @param callback    Callback method
 */
public static void setSoundModeConfig(String sn, boolean silentMode, boolean
voiceEnable, MNOpenSDKInterface.SetSoundModeConfigCallBack callback);
```

2.12 获取设备时区配置（Get device time zone configuration)

```
/**
 * 获取设备时区配置（Get device time zone configuration）
 *
 * @param sn       Device sn
 * @param callback Callback method
 */
public static void getTimeZoneConfig(String sn,
MNOpenSDKInterface.GetTimeZoneConfigCallBack callback);
```

2.13设置设备时区配置（Set device time zone configuration)

```
/**
 * 设置设备时区配置（Set device time zone configuration）
 *
 * @param sn       Device sn
 * @param timeZone 时区对照id（Time zone control id）
 * @param callback Callback method
 */
public static void setTimeZoneConfig(String sn, int timeZone,
MNOpenSDKInterface.SetTimeZoneConfigCallBack callback);
```

2.14 获取设备夏令时配置（Get device daylight saving time configuration)

```
/**
 * 获取设备夏令时配置（Get device daylight saving time configuration）
 *
 * @param sn       Device sn
 * @param callback Callback method
 */
public static void getSummerTimeConfig(String sn,
MNOpenSDKInterface.GetSummerTimeConfigCallBack callback);
```

2.15 设置设备夏令时配置（Set device daylight saving time configuration）

```
/**
 * 设置设备夏令时配置（Set device daylight saving time configuration）
 *
 * @param sn         Device sn
 * @param DSTEnable 是否开启夏令时（Whether to enable daylight saving time）
 * @param callback   Callback method
 */
public static void setSummerTimeConfig(String sn, boolean DSTEnable,
MNOpenSDKInterface.SetSummerTimeConfigCallBack callback);
```

2.16 获取设备呼吸灯配置（Get device breathing light configuration）

```
/**
 * 获取设备呼吸灯配置（Get device breathing light configuration）
 *
 * @param sn       Device sn
 * @param callback Callback method
 */
public static void getNetLightConfig(String sn,
MNOpenSDKInterface.GetNetLightCallBack callback);
```

2.17 设置设备呼吸灯配置（Set device breathing light configuration）

```
/**
 * 设置设备呼吸灯配置（Set device breathing light configuration）
 *
 * @param sn       Device sn
 * @param netLight 是否开启呼吸灯（Whether to turn on the breathing light）
 * @param callback Callback method
 */
public static void setNetLightConfig(String sn, boolean netLight,
MNOpenSDKInterface.SetNetLightCallBack callback);
```

2.18 获取背光补偿配置(Get gun camera backlight compensation configuration)

```
/**
 * 获取背光补偿配置(Get gun camera backlight compensation configuration)
 *
 * @param sn       Device sn
 * @param callback Callback method
 */
public static void getBLCConfig(String sn,
MNOpenSDKInterface.GetBLCConfigCallBack callback);
```

2.19 设置背光补偿开启与关闭状态(Setting backlight compensation on and off)

```
/**
 * 设置背光补偿开启与关闭状态(Setting backlight compensation on and off)
 *
 * @param sn              Device sn
 * @param LightEnable     是否开启背光补偿（Whether to enable backlight
compensation）
 * @param LightSensitive 光敏值（Photosensitivity）
 * @param callback        Callback method
 */
public static void setBLCConfig(String sn, boolean LightEnable, int
LightSensitive, MNOpenSDKInterface.SetBLCConfigCallBack callback);
```

2.20获取设备红外灯、镜像、翻转配置(Get device infrared light, mirror, flip configuration)

```
/**
 * 获取设备红外灯、镜像、翻转配置(Get device infrared light, mirror, flip
configuration)
 *
 * @param sn       Device sn
 * @param callback Callback method
 */
public static void getVideoInOptions(String sn,
MNOpenSDKInterface.GetVideoInOptionsCallBack callback);
```

2.21 设置设备红外灯、镜像、翻转配置(Set device infrared light, mirror, flip configuration)

```
/**
 * 设置设备红外灯、镜像、翻转配置(Set device infrared light, mirror, flip
configuration)
 *
 * @param sn        Device sn
 * @param inOptBean 配置信息(Configuration information)
 * @param callback  Callback method
 */
public static void setVideoInOptions(String sn, VideoInOptBean inOptBean,
MNOpenSDKInterface.SetVideoInOptionsCallBack callback);
```

2.22 获取NVR对应通道设备红外灯、镜像、翻转配置(Get NVR corresponding channel device infrared light, mirror, flip configuration)

```
/**
 * 获取NVR对应通道设备红外灯、镜像、翻转配置(Get NVR corresponding channel device
infrared light, mirror, flip configuration)
 *
 * @param sn       Device sn
 * @param channels 通道号(Channel number)
 * @param callback Callback method
 */
public static void getNvrVideoInOptions(String sn, int[] channels,
MNOpenSDKInterface.GetNvrVideoInOptionsCallBack callback);
```

2.23 设置NVR对应通道设备设备红外灯、镜像、翻转配置(Set NVR corresponding channel device equipment infrared light, mirror, flip configuration)

```
/**
 * 设置NVR对应通道设备设备红外灯、镜像、翻转配置(Set NVR corresponding channel device
equipment infrared light, mirror, flip configuration)
 *
 * @param sn         Device sn
 * @param inOptBeans 配置信息(Configuration information)
 * @param callback   Callback method
 */
public static void setNvrVideoInOptions(String sn, ArrayList<VideoInOptBean>
inOptBeans, MNOpenSDKInterface.SetNvrVideoInOptionsCallBack callback);
```

2.24 获取声光模式设置(Get sound and light mode settings)

```
/**
 * 获取声光模式设置(Get sound and light mode settings)
 *
 * @param sn       Device sn
 * @param callback Callback method
 */
public static void getAlarmAsossiatedConfig(String sn,
MNOpenSDKInterface.GetAlarmAsossiatedConfigCallBack callback);
```

2.25 设置声光模式(Set acousto-optic mode)

```
/**
 * 设置声光模式(Set acousto-optic mode)
 *
 * @param sn          Device sn
 * @param LightType   白光灯是否开启(Whether the white light is on)
 * @param audioEnable 警示音是否开启(Whether the warning sound is on)
 * @param callback    Callback method
 */
public static void setAlarmAsossiatedConfig(String sn, int LightType, boolean
audioEnable, MNOpenSDKInterface.SetAlarmAsossiatedConfigCallBack callback);
```

2.26 获取设备动检使能和灵敏度配置(Get device dynamic detection enable and sensitivity configuration)

```
/**
 * 获取设备动检使能和灵敏度配置(Get device dynamic detection enable and sensitivity
configuration)
 *
 * @param sn       Device sn
 * @param callback Callback method
 */
public static void getMotionDetectConfig(String sn,
MNOpenSDKInterface.GetMotionDetectCallBack callback);
```

2.27 设置设备动检使能和灵敏度配置(Set device motion detection enable and sensitivity configuration)

```
/**
 * 设置设备动检使能和灵敏度配置(Set device motion detection enable and sensitivity
configuration)
 *
 * @param sn           Device sn
 * @param sensitivity  移动侦测灵敏度(Motion detection sensitivity)
 * @param motionDetect 移动侦测开关(Motion detection switch)
 * @param callback     Callback method
 */
public static void setMotionDetectConfig(String sn, int sensitivity, boolean
motionDetect, MNOpenSDKInterface.SetMotionDetectCallBack callback);
```

2.28 获取NVR设备动检使能和灵敏度配置(Get NVR device motion detection enable and sensitivity configuration)

```
/**
 * 获取NVR设备动检使能和灵敏度配置(Get NVR device motion detection enable and
sensitivity configuration)
 *
 * @param sn       Device sn
 * @param channels 设备通道号 从0开始(Device channel number starting from 0)
 * @param callback Callback method
 */
public static void getNVRMotionDetectConfig(String sn, int[] channels,
MNOpenSDKInterface.GetNVRMotionDetectCallBack callback);
```

2.29 设置NVR设备动检使能和灵敏度配置(Set NVR device motion detection enable and sensitivity configuration)

```
/**
 * 设置NVR设备动检使能和灵敏度配置(Set NVR device motion detection enable and
sensitivity configuration)
 *
 * @param sn            Device sn
 * @param motionDetects 对应通道设置信息(Corresponding channel setting information)
 * @param callback      Callback method
 */
public static void setNVRMotionDetectConfig(String sn, ArrayList<MotionDetect>
motionDetects, MNOpenSDKInterface.SetNVRMotionDetectCallBack callback);
```

2.30 获取设备人脸识别使能配置(Get device face recognition enabled configuration)

```
/**
 * 获取设备人脸识别使能配置(Get device face recognition enabled configuration)
 *
 * @param sn       Device sn
 * @param callback Callback method
 */
public static void getFaceDetectConfig(String sn,
MNOpenSDKInterface.GetFaceDetectCallBack callback);
```

2.31 设置设备人脸识别使能配置(Set device face recognition enable configuration)

```
/**
 * 设置设备人脸识别使能配置(Set device face recognition enable configuration)
 *
 * @param sn            Device sn
 * @param faceDetection 是否开启人脸识别（Whether to enable face recognition）
 * @param callback      Callback method
 */
public static void setFaceDetectConfig(String sn, boolean faceDetection,
MNOpenSDKInterface.SetFaceDetectCallBack callback);
```

2.32 获取NVR对应设备人脸识别配置（Get the face recognition configuration of the NVR corresponding device)

```
/**
 * 获取NVR对应设备人脸识别配置（Get the face recognition configuration of the NVR
corresponding device）
 *
 * @param sn       Device sn
 * @param channels 设备通道号（Device channel number）
 * @param callback Callback method
 */
public static void getNvrFaceDetectConfig(String sn, int[] channels,
MNOpenSDKInterface.GetNvrFaceDetectCallBack callback);
```

2.33 设置NVR设备对应通道人脸识别状态（Set the face recognition status of the corresponding channel of the NVR device)

```
/**
 * 设置NVR设备对应通道人脸识别状态（Set the face recognition status of the
corresponding channel of the NVR device）
 *
 * @param sn        Device sn
 * @param faceBeans 对应通道的人脸识别开关配置（Face recognition switch configuration
for the corresponding channel）
 * @param callback  Callback method
 */
public static void setNvrFaceDetectConfig(String sn, ArrayList<FaceBean>
faceBeans, MNOpenSDKInterface.SetNvrFaceDetectCallBack callback);
```

2.34 获取P2摇头机设备移动追踪开关（Get P2 shaker device movement tracking switch)

```
/**
 * 获取P2摇头机设备移动追踪开关（Get P2 shaker device movement tracking switch）
 *
 * @param sn       Device sn
 * @param callback Callback method
 */
public static void getMotionTrackConfig(String sn,
MNOpenSDKInterface.GetMotionTrackConfigCallBack callback);
```

2.35 设置P2摇头机设备移动追踪开关（Set the P2 shaker device movement tracking switch)

```
/**
 * 设置P2摇头机设备移动追踪开关（Set the P2 shaker device movement tracking switch）
 *
 * @param sn          Device sn
 * @param motionTrack 是否开启移动追踪（Whether to enable mobile tracking）
 * @param callback    Callback method
 */
public static void setMotionTrackConfig(String sn, boolean motionTrack,
MNOpenSDKInterface.SetMotionTrackConfigCallBack callback);
```

2.36 获取设备视频是否上传云端和视频长度配置（Get whether device video is uploaded to the cloud and video length configuration)

```
/**
 * 获取设备视频是否上传云端和视频长度配置（Get whether device video is uploaded to the
cloud and video length configuration）
 *
 * @param sn       Device sn
 * @param callback Callback method
 */
public static void getAlarmCloudRecordConfig(String sn,
MNOpenSDKInterface.GetAlarmCloudRecordConfigCallBack callback);
```

2.37 设置设备视频是否上传云端和视频长度配置（Set whether device video is uploaded to the cloud and video length configuration)

```
/**
 * 设置设备视频是否上传云端和视频长度配置（Set whether device video is uploaded to the cloud and video length configuration）
 *
 * @param sn         Device sn
 * @param nvrSetBean 配置信息（Configuration information）
 * @param callback   Callback method
 */
public static void setAlarmCloudRecordConfig(String sn, AlarmCloudRecordSetBean nvrSetBean, MNOpenSDKInterface.SetAlarmCloudRecordConfigCallBack callback);
```

2.38 截取设置当前画面（Capture the current picture)

```
/**
 * 截取设置当前画面（Capture the current picture）
 *
 * @param sn       Device sn
 * @param callback Callback method
 */
public static void getCapturePicture(String sn, MNOpenSDKInterface.GetCapturePictureCallBack callback);
```

2.39 获取NVR设备信息(Get NVR Device Information)

```
/**
 * 获取NVR设备信息(Get NVR Device Information)
 *
 * @param sn       Device sn
 * @param callback Callback method
 */
public static void getNVRIPCInfo(String sn, MNOpenSDKInterface.GetNVRIPCInfoCallBack callback);
```

2.40 低功耗电池设备保活（Low power battery equipment keep alive)

```
/**
 * 低功耗电池设备保活（Low power battery equipment keep alive）
 *
 * @param sn Device sn
 */
public static void setKeepAlive(String sn);
```

2.41 设备获取设备固件升级进度（Device gets device firmware upgrade progress)

```
/**
 * 设备获取设备固件升级进度（Device gets device firmware upgrade progress）
 *
 * @param sn Device sn
 */
public static void getUpgradeState(String sn,
MNOpenSDKInterface.GetUpgradeStateCallBack callback);
```

2.42 设备固件升级（Device firmware upgrade)

```
/**
 * 设备固件升级（Device firmware upgrade）
 *
 * @param sn        Device sn
 * @param pPkgUrl
 * @param lPkgSize
 * @param pszPkgMD5
 */
public static void upgradeFirmware(String sn, String pPkgUrl, long lPkgSize,
String pszPkgMD5);
```

2.43 重启设备（reboot device)

```
/**
 * 重启设备（reboot device）
 *
 * @param sn Device sn
 */
public static void rebootDevice(String sn);
```

2.44 重置设备wifi（Reset device wifi)

```
/**
 * 重置设备wifi（Reset device wifi）
 *
 * @param sn Device sn
 */
public static void restoreWLan(String sn);
```

3 存储卡配置

3.1 获取TF卡名称、是否需要格式化、总容量、剩余容量(Get TF card name, whether formatting is required, total capacity, remaining capacity)

```
/**
 * 获取TF卡名称、是否需要格式化、总容量、剩余容量(Get TF card name, whether formatting
is required, total capacity, remaining capacity)
 *
 * @param sn       Device sn
 * @param callback Callback method
 */
public static void getTFStateConfig(String sn,
MNOpenSDKInterface.GetTFStorageCallBack callback);
```

3.2 获取TF卡名称、是否需要格式化(Get TF card name, whether it needs to be formatted)

```
/**
 * 获取TF卡名称、是否需要格式化(Get TF card name, whether it needs to be formatted)
 *
 * @param sn       Device sn
 * @param callback Callback method
 */
public static void getTFSimpleState(String sn,
MNOpenSDKInterface.GetTFSimpleStateCallBack callback);
```

3.3 设备TF卡格式化(Device TF card format)

```
/**
 * 设备TF卡格式化(Device TF card format)
 *
 * @param sn       Device sn
 * @param tfName   要格式化的卡名称(Card name to format)
 * @param callback Callback method
 */
public static void setTFStorageFormatting(String sn, String tfName,
MNOpenSDKInterface.TFStorageFormatCallBack callback);
```

3.4 获取设备TF卡存储类型和时间配置(Get device TF card storage type and time configuration)

```
/**
 * 获取设备TF卡存储类型和时间配置(Get device TF card storage type and time
configuration)
 *
 * @param sn       Device sn
 * @param callback Callback method
 */
public static void getAlarmRecord(String sn,
MNOpenSDKInterface.GetAlarmRecordCallBack callback);
```

3.5 设置设备TF卡存储类型和时间配置(Set device TF card storage type and time configuration)

```
/**
 * 设置设备TF卡存储类型和时间配置(Set device TF card storage type and time
configuration)
 *
 * @param sn             Device sn
 * @param isAllDayRecord 是否开启全时录像(Whether to enable full-time recording)
 * @param callback       Callback method
 */
public static void setAlarmRecord(String sn, boolean isAllDayRecord,
MNOpenSDKInterface.SetAlarmRecordCallBack callback);
```

3.6 获取NVR设备TF卡存储类型和时间配置(Get device TF card storage type and time configuration)

```
/**
 * 获取设备TF卡存储类型和时间配置(Get device TF card storage type and time
configuration)
 *
 * @param sn       Device sn
 * @param channels 通道号(Channel number)
 * @param callback Callback method
 */
public static void getNvrAlarmRecord(String sn, int[] channels,
MNOpenSDKInterface.GetNvrAlarmRecordCallBack callback);
```

3.7 设置NVR设备TF卡存储类型和时间配置(Set device TF card storage type and time configuration)

```
/**
 * 设置设备TF卡存储类型和时间配置(Set device TF card storage type and time
configuration)
 *
 * @param sn           Device sn
 * @param alarmRecords 设置对应通道是否开启全时录像的配置(Configure whether the
corresponding channel is enabled for full-time recording)
 * @param callback     Callback method
 */
public static void setNvrAlarmRecord(String sn, ArrayList<SetAlarmRecordBean>
alarmRecords, MNOpenSDKInterface.SetNvrAlarmRecordCallBack callback);
```

4 传感器设备

4.1 向传感器发送wifi信息（Send wifi information to the sensor)

```
/**
 * 向传感器发送wifi信息（Send wifi information to the sensor）
 *
 * @param ssid     wifi名称（wifi name）
 * @param pwd      wifi密码（wifi password）
 * @param pwdMode  加密方式 例如：WPA2-AES （Encryption method Example: WPA2-AES）
 * @param userID   用户ID（User ID）
 * @param areaCode 区域码，无此字段则认为空（Area code, without this field it is considered empty）
 * @param domain   域名附加段，例如"in"，则对应域名（Additional domain name segment, such as "in", corresponding to the domain name）iotin.bullyun.com ，videoin.bullyun.com
 * @return 0发送成功，-1创建套接字失败，-2连接服务器失败，-3发送wifi信息失败,-4参数有问题（0 sent successfully, -1 failed to create a socket, -2 failed to connect to the server, -3 failed to send wifi information, -4 has a problem with the parameter）
 */
public static int SendWifiConfig(String ssid, String pwd, String pwdMode, String userID, String areaCode, String domain);
```

4.2 传感器绑定请求（Sensor binding request)

```
/**
 * APP收到传感器绑定的请求后需要调用此接口（APP needs to call this interface after receiving the sensor binding request）
 *
 * @param sn       Device sn
 * @param response JSON格式的响应,{"bindUser":{"name":"zhangsan"}}
 * @param code     0 绑定成功，-1 绑定失败（0 binding succeeded, -1 failed to bind）
 * @return 0 发送成功，-1 发送失败（0 sent successfully, -1 sent failed）
 */
public static int ResponseBindDevice(String sn, String response, int code);
```